

IBM External Submission #24167

Reprint Courtesy of International Business Machines Corporation, © 2015 International Business Machines Corporation

INTERNATIONAL BUSINESS MACHINES CORPORATION (IBM) ARMONK, NEW YORK 10504

PERMISSION TO REPRINT IBM COPYRIGHTED PUBLICATIONS

Each reprint must be accompanied by the following credit line: **“Reprint Courtesy of International Business Machines Corporation, © (year) International Business Machines Corporation”**. The credit line normally should appear on the page where the reprint appears, either under the title or as a footnote.

If the foregoing is inconvenient, the credit line may be placed on the face or back of the title page (or front cover, if there is no title page) or in a conveniently viewable manner with suitable reference to the place where the reprint appears.

When multiple IBM materials are reprinted in the same publication, a consolidated credit paragraph may be used on the title page, or in a conveniently viewable manner listing the titles, corresponding copyright notices and references to the points where the reprints appear.

It is the understanding of **International Business Machines Corporation** that the purpose for which its material is being used is accurate and true as stated in the original request.

Permission to quote from, transmit electronically or reprint IBM material is limited to the purpose and quantities originally requested and must not be construed as a blanket license to use the material for other purposes or to reprint other IBM copyrighted material.

IBM reserves the right to withdraw permission to use copyrighted material whenever, in its discretion, it feels that the privilege of using its material is being used in a way detrimental to its interest or the above instructions are not being followed properly to protect its copyright.

No permission is granted to use trademarks of **International Business Machines Corporation** and its affiliates apart from the incidental appearance of such trademarks in the titles, text, and illustrations of the named publications. Any proposed use of trademarks apart from such incidental appearance requires separate approval in writing and ordinarily cannot be given. The use of any IBM trademark should not be of a manner which might cause confusion of origin or appear to endorse non-IBM products.

THIS PERMISSION IS PROVIDED WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

INTERNATIONAL BUSINESS MACHINES CORPORATION

Dated: August 19, 2015

Presentation Manager Programming Guide and Reference

IBM OS/2 Toolkit Documentation

How to Use this Book

This reference is a detailed technical guide and reference for application programmers. It gives reference information and code examples to enable you to write source code using Presentation Manager functions.

Before you begin to use this information, it would be helpful to understand how you can:

- Expand the Contents to see all available topics
- Obtain additional information for a highlighted word or phrase
- Use action bar choices
- Use the programming information

How to Use the Contents

When the Contents window first appears, some topics have a plus (+) sign beside them. The plus sign indicates that additional topics are available.

To expand the Contents if you are using a mouse, click on the plus sign. If you are using the keyboard, use the Up or Down Arrow key to highlight the topic, and press the plus (+) key. For example, **Code Pages** has a plus sign beside it. To see additional topics for that heading, click on the plus sign or highlight that topic and press the plus (+) key.

To view a topic, double-click on the topic (or press the Up or Down Arrow key to highlight the topic, and then press the Enter key).

How to Obtain Additional Information

After you select a topic, the information for that topic appears in a window. Highlighted words or phrases indicate that additional information is available. You will notice that certain words and phrases are highlighted in green letters, or in white letters on a black background. These are called hypertext terms. If you are using a mouse, double-click on the highlighted word. If you are using a keyboard, press the Tab key to move to the highlighted word, and then press the Enter key. Additional information then appears in a window.

How to Use Action Bar Choices

Several choices are available for managing information presented in the *Presentation Manager Programming Reference*. There are three pull-down menus on the action bar: the **Services** menu, the **Options** menu, and the **Help** menu.

The actions that are selectable from the **Services** menu operate on the active window currently displayed on the screen. These actions include the following:

Bookmark

Allows you to set a placeholder so you can retrieve information of interest to you.

When you place a bookmark on a topic, it is added to a list of bookmarks you have previously set. You can view the list, and you can remove one or all bookmarks from the list. If you have not set any bookmarks, the list is empty.

To set a bookmark, do the following:

1. Select a topic from the Contents.
2. When that topic appears, choose the **Bookmark** option from the **Services** pull-down.
3. If you want to change the name used for the bookmark, type the new name in the field.
4. Click on the **Place** radio button (or press the Up or Down Arrow key to select it).
5. Click on **OK** (or select it and press Enter). The bookmark is then added to the bookmark list.

Search

Allows you to find occurrences of a word or phrase in the current topic, selected topics, or all topics.

You can specify a word or phrase to be searched. You can also limit the search to a set of topics by first marking the topics in the Contents list.

To search for a word or phrase in all topics, do the following:

1. Choose the **Search** option from the **Services** pull-down.
2. Type the word or words to be searched for.
3. Click on **All sections** (or press the Up or Down Arrow keys to select it).

4. Click on **Search** (or select it and press Enter) to begin the search.
5. The list of topics where the word or phrase appears is displayed.

Print

Allows you to print one or more topics. You can also print a set of topics by first marking the topics in the Contents list.

To print the document Contents list, do the following:

1. Choose **Print** from the **Services** pull-down.
2. Click on **Contents** (or press the Up or Down Arrow key to select it).
3. Click on **Print** (or select it and press Enter).
4. The Contents list is printed on your printer.

Copy

Allows you to copy a topic that you are viewing to the System Clipboard or to a file that you can edit. You will find this particularly useful for copying syntax definitions and program samples into the application that you are developing.

You can copy a topic that you are viewing in two ways:

- **Copy** copies the topic that you are viewing into the System Clipboard. If you are using a Presentation Manager editor (for example, the System Editor) that copies or cuts (or both) to the System Clipboard, and pastes to the System Clipboard, you can easily add the copied information to your program source module.
- **Copy to file** copies the topic that you are viewing into a temporary file named TEXT.TMP. You can later edit that file by using any editor. You will find TEXT.TMP in the directory where your viewable document resides.

To copy a topic, do the following:

1. Expand the Contents list and select a topic.
2. When the topic appears, choose **Copy to file** from the **Services** pull-down.
3. The system puts the text pertaining to that topic into the temporary file named TEXT.TMP.

For information on one of the other choices in the **Services** pull-down, highlight the choice and press the F1 key.

The actions that are selectable from the **Options** menu allow you to change the way your Contents list is displayed. To expand the Contents and show all levels for all topics, choose **Expand all** from the **Options** pull-down. You can also press the Ctrl and * keys together. For information on one of the other choices in the **Options** pull-down, highlight the choice and press the F1 key.

The actions that are selectable from the **Help** menu allow you to select different types of help information. You can also press the F1 key for help information about the Information Presentation Facility (IPF).

How to Use the Programming Information

This document consists of guide and reference information that provides a detailed description of each function, message, constant, and data type. It provides language-dependent information about the functions which enable the user to generate call statements in the C Language.

Presentation Manager programming information is presented by component, such as Device Functions, Direct Manipulation Functions, and Font-File format, for example:

Contents

- + Device Functions
- + Direct Manipulation Functions
- + Font-File Format

By clicking on the plus sign beside "Device Functions", you see an alphabetic list of the Presentation Manager Device functions. Selecting a function takes you directly into the reference information for that function.

Units of reference information are presented in selectable multiple windows or viewports. A viewport is a Presentation Manager window that can be sized, moved, minimized, maximized, or closed. By selecting a unit (in this case, an entry on the Contents list), you will see two windows displayed:

Unit Title	Selection Title
Select an item	
Syntax	
Returns	
Notes	
Example Code	
Related Functions	
Glossary	

The window on the left is the primary window. It contains a list of items that are always available to you. The window on the right is the secondary window. It contains a "snapshot" of the unit information. For reference units (that is, function descriptions), this window contains the Function Syntax.

All of the information needed to understand a reference unit (or topic) is readily available to you through the primary window. The information is divided into discrete information groups, and only the appropriate information group appears for the topic that you are viewing.

The information groups for a reference unit (that is, a function description) can include all or some of the following:

- Syntax
- Parameters
- Returns
- Errors
- Notes
- Example Code
- Related Functions
- Graphic Elements and Orders
- Glossary

This list may vary. Some topics may be omitted when they do not apply.

Information groups are displayed in separate viewports that are stacked in a third window location that overlaps the secondary window. By selecting an item (information group) in the primary window, the item is displayed in the third window location, as follows:

Unit Title	Selection	Glossary
Select an item		Select a starting letter of glossary terms
.		
.		
.	A	N
.	B	O
.	C	P
Glossary	.	.
	.	.
	.	.
	M	Z

By selecting successive items from the primary window, additional windows are displayed on top of the previous windows displayed in the third window location. For example, in a function description, Parameters and Return Values are items listed in the primary window. When selected, they appear one on top of the other in the third window location. Because of this, you may move the first selected (topmost) window to the left before selecting the next item. This allows simultaneous display of two related pieces of information from the "stack" of windows in the third window location, as follows:

Unit Title	Parameters	Return Values
Select an item		
.		
.		
.		
Returns		
Errors		
.		
.		
.		

Each window can be individually closed from its system menu. All windows are closed when you close the primary window.

Some secondary windows may have the appearance of a split screen. For example, an illustration may appear in the left half of the window, and scrollable, explanatory information may appear in the right half of the window. Because illustrations may not necessarily fit into the small window size on your screen, you may maximize the secondary window for better readability.

Conventions Used in this Book

The purpose of this reference is to give important information about functions, messages, constants, and data types. It provides language-dependent information about the functions which enables the user to call functions in the C programming language.

The following information is provided:

- The syntax and parameters for each function
- The syntax of each data type and structure

Notation Conventions

The following notation conventions are used in this reference:

NULL	The term NULL applied to a parameter is used to indicate the presence of the pointer parameter, but with no value.										
NULLHANDLE	The term NULLHANDLE applied to a parameter is used to indicate the presence of the handle parameter, but with no value.										
Implicit Pointer	If no entry for a data type "Pxxxxxx" is found in then it is implicitly a pointer to the data type "xxxxxx". See Implicit Pointer Data Types for more information about implicit pointers.										
Constant Names	<p>All constants are written in uppercase to match the header files. Where applicable, constant names have a prefix derived from the name of a function, message, or idea associated with the constant. For example:</p> <table><tr><td>WM_CREATE</td><td>Window message</td></tr><tr><td>SV_CXICON</td><td>System value</td></tr><tr><td>CF_TEXT</td><td>Clipboard format.</td></tr></table> <p>In this book, references to a complete set of constants with a given prefix is written as shown in the following examples:</p> <table><tr><td>Window message</td><td>WM_*</td></tr><tr><td>System value</td><td>SV_*</td></tr></table>	WM_CREATE	Window message	SV_CXICON	System value	CF_TEXT	Clipboard format.	Window message	WM_*	System value	SV_*
WM_CREATE	Window message										
SV_CXICON	System value										
CF_TEXT	Clipboard format.										
Window message	WM_*										
System value	SV_*										
Parameters and Fields	Function parameters and data structure fields are shown in italics.										

Conventions Used in Function Descriptions

The documentation of each function contains these sections:

Syntax

The function syntax describes the C-language calling syntax of the function and gives a brief description.

Programming Note

The functions in this book are spelled in mixed-case for readability but are known to the system as uppercase character strings. For example, the function "GPIBeginArea" is actually the external name "GPIBEGINAREA".

If you are using a compiler that generates a mixed-case external name, you should code the functions in uppercase.

Parameters

Each parameter is listed with its C-language data type, parameter type, and a brief description.

- All data types are written in uppercase to match the header files. A data type of "Pxxxxxxx" implicitly defines a pointer to the data type "xxxxxxx".

The term NULL applied to a parameter indicates the presence of the parameter, with no value.

Refer to for a complete list of all data types and their descriptions.

- There are three parameter types:

Input	Specified by the programmer.
Output	Returned by the function.
Input/Output	Specified by the programmer and modified by the function.

- A brief description is provided with each parameter. Where appropriate, restrictions are also included. In some cases, the parameter points to a structure.

Returns

A list of possible return codes or errors (when appropriate) is included in this section. Some functions do not have return codes. Refer to for a list of error codes and their numerical values, and for a list of error codes and their descriptions.

For some functions, this section includes a statement that the function requires a message queue. This means that, before issuing a call, [WinCreateMsgQueue](#) must be issued by the same thread. For other functions, no previous [WinCreateMsgQueue](#) is required, and it is only necessary to issue [WinInitialize](#) from the same thread.

Remarks

This section contains additional information about the function, when required.

Related Functions

This list shows the functions (if any) that are related to the function being described.

Example Code

An example of how the function can be used is shown in C language.

Error Severities

Each of the error conditions given in the list of errors for each function falls into one of these areas:

Warning

The function detected a problem, but took some remedial action that enabled the function to complete successfully. The return code in this case indicates that the function completed successfully.

Error

The function detected a problem for which it could not take any sensible remedial action. The system has recovered from the problem, and the state of the system, with respect to the application, remains the same as at the time when the function was requested. The system has not even partially executed the function (other than reporting the error).

Severe Error

The function detected a problem from which the system could not reestablish its state, with respect to the application, at the time when that function was requested; that is, the system partially executed the function. This necessitates the application performing some corrective activity to restore the system to some known state.

Unrecoverable Error

The function detected some problem from which the system could not re-establish its state, with respect to the application, at the time when that call was issued. It is possible that the application cannot perform some corrective action to restore the system to some

known state.

The [WinGetLastError](#) and [WinGetErrorInfo](#) functions can be used to find out more about an error (or warning) that occurs as a result of executing a call.

Header Files

All functions require an "#include" statement for the system header file OS2.H:

```
#include <OS2.H>
```

Most functions also require a "#define" statement to select an appropriate (conditional) section of the header file, and hence, the required prototype. Where this is necessary, it is shown at the head of the function definition in the form:

```
#define INCL_name
```

Note: These "#define" statements must precede the "#include <OS2.H>" statement.

Helper Macros

A series of macros is defined for packing data into, and extracting data from, variables of [MPARAM](#) and [MRESULT](#) data types. They are used in conjunction with the [WinSendMsg](#) and the other message functions, and also inside window and dialog procedures.

These macros always cast their arguments to the specified type, so values of any of the types specified for each macro can be passed without additional casting. NULL can be used to pass unused parameter data.

Macros for packing data into a [MPARAM](#) variable are shown below:

```
/* Used to pass any pointer type: */
#define MPFROMP(p) ((MPARAM)(VOID*)(p))

/* Used to pass a window handle: */
#define MPFROMHWND(hwnd) ((MPARAM)(HWND)(hwnd))

/* Used to pass a CHAR, UCHAR, or BYTE: */
#define MPFROMCHAR(ch) ((MPARAM)(USHORT)(ch))

/* Used to pass a LONG, ULONG, or BOOL: */
#define MPFROMLONG(l) ((MPARAM)(ULONG)(l))

/* Used to pass two SHORTs or USHORTs: */
#define MPFROM2SHORT(s1, s2) ((MPARAM)MAKELONG(s1, s2))

/* Used to pass a SHORT and 2 UCHARs: (WM_CHAR msg)*/
#define MPFROMSH2CH(s, uch1, uch2) ((MPARAM)MAKELONG(s, MAKESHORT(uch1, uch2)))
```

Macros for extracting data from a [MPARAM](#) variable are shown below:

```
/* Used to get any pointer type: */
#define PVOIDFROMMP(mp) ((VOID*)(mp))

/* Used to get a window handle: */
```



```

#define HWNDFROMMP(mp)      ((HWND)(mp))

/* Used to get CHAR, UCHAR, or BYTE: */
#define CHAR1FROMMP(mp)     ((UCHAR)(mp))
#define CHAR2FROMMP(mp)     ((UCHAR)((ULONG)mp >> 8))
#define CHAR3FROMMP(mp)     ((UCHAR)((ULONG)mp >> 16))
#define CHAR4FROMMP(mp)     ((UCHAR)((ULONG)mp >> 24))

/* Used to get a LONG, ULONG, or BOOL: */
#define LONGFROMMP(mp)      ((ULONG)(mp))

```

Macros for packing data into a [MRESULT](#) variable are shown below:

```

/* Used to pass any pointer type: */
#define MRFROMP(p)           ((MRESULT)(VOID*)(p))

/* Used to pass a LONG, ULONG, or BOOL: */
#define MRFROMLONG(l)        ((MRESULT)(ULONG)(l))

/* Used to pass two SHORTs or USHORTs: */
#define MRFROM2SHORT(s1, s2) ((MRESULT)MAKELONG(s1, s2))

```

Macros for extracting data from a [MRESULT](#) variable are shown below:

```

/* Used to get any pointer type: */
#define PVOIDFROMMR(mr)      ((VOID*)(mr))

/* Used to get a LONG, ULONG, or BOOL: */
#define LONGFROMMR(mr)       ((ULONG)(mr))

```

The following macros are for use with [DDESTRUCT](#) and [DDEINIT](#) structures are shown below:

```

/* Used to return a PSZ pointing to the DDE item name: */
#define DDES_PSZITEMNAME(pddes) \
    (((PSZ)pddes) + ((PDDESTRUCT)pddes)->offsItemName)

/* Used to return a PBYTE pointing to the DDE data: */
#define DDES_PABDATA(pddes) \
    (((PBYTE)pddes) + ((PDDESTRUCT)pddes)->offabData)

/* Used to convert a selector to a PDDESTRUCT: */
#define SELTOPDDES(sel)       ((PDDESTRUCT)MAKEP(sel, 0))

/* Used to PDDESTRUCT to a selector for freeing / reallocating: */
#define PDDESTOSEL(pddes)     (SELECTOROF(pddes))

/* Used to PDDEINIT to a selector for freeing: */
#define PDDEITOSEL(pddei)     (SELECTOROF(pddei))

```

Addressing Elements in Arrays

Constants defining array elements are given values that are zero-based in C; that is, the numbering of the array elements starts at zero, not one.

For example, in the [DevQueryCaps](#) function, the sixth element of the *alArray* parameter is CAPS_HEIGHT, which is equated to 5.

Count parameters related to such arrays always mean the actual number of elements available; therefore, again using the [DevQueryCaps](#) function as an example, if all elements up to and including CAPS_HEIGHT are provided for, *lCount* could be set to (CAPS_HEIGHT+1).

In functions for which the starting array element can be specified, this is always zero-based, and so the C element number constants can be

used directly. For example, to start with the CAPS_HEIGHT element, the */Start* parameter can be set to CAPS_HEIGHT.

Implicit Pointer Data Types

A data type name beginning with "P" (for example, PERRORCODE) is likely to be a pointer to another data type (in this instance, ERRORCODE).

In the data type summary, no explicit "typedefs" are shown for pointers; therefore, if no data type definition can be found in the summary for a data type name "Pxxxxxx", it represents a pointer to the data type "xxxxxx", for which a definition should be found in the reference.

The implicit type definition needed for such a pointer "Pxxxxxx" is:

```
typedef xxxxxx *Pxxxxxx;
```

Such definitions are provided in the header files.

Storage Mapping of Data Types

The storage mapping of the data types is dependent on the machine architecture. To be portable, applications must access the data types using the definitions supplied for the environment in which they will execute.

Double-Byte Character Set (DBCS)

Throughout this publication, you will see references to specific value for character strings. The values are for single-byte character set (SBCS). If you use the double-byte character set (DBCS), note that one DBCS character equals two SBCS characters.

Programming Considerations

This section provides information you need to consider before you begin programming with Presentation Manager functions.

Stack Size

Existing 16-bit applications (small and tiny models) must have a 4KB stack available when they enter system calls; otherwise, the stack can overflow into the data area.

Presentation Manager

The Presentation Manager component of the OS/2 operating system is based on the IBM Systems Application Architecture (SAA) Common

Programming Interface-a an architecture for the design and development of applications.

The Presentation Manager component implements the Common User Access (CUA) interface, which you can use to attain consistency in the appearance and behavior of your applications.

C++ Considerations

This section contains several topics you should take into consideration if you are using C++ to develop applications.

A Few Words on Typedefs

The OS/2 header files can be used with either C or C++ compilers. If `__cplusplus` has been defined, the header files will produce code that is compatible with C++. This is done since several of the typedefs have been changed to support C++. For example, many items that are unsigned char in the "C header files" are char when compiled with `__cplusplus`. For instance,

```
typedef unsigned char BYTE;
```

has changed to

```
typedef char BYTE;
```

The existing samples that are included in the IBM Developer's Toolkit for OS/2 Warp Version 3 can be compiled with either C or C++.

Note: Prior versions of the IBM Developer's Toolkit for OS/2 Warp Version 3 provided two sets of header files: one set for use with C compilers, another set for use with C++ compilers. The same set of header files are now used by either compiler.

PCSZ Data Type

If a function takes as a parameter a string that is not changed by the function, the string parameter can be declared as a "const" string, or a [PCSZ](#). [PCSZ](#) is defined in the header files as a "const" pointer to a NULL-delimited string.

The "const" means that the function will not change the contents of the string. The use of the "const" keyword is not specific to C++, many C compilers support it as well. However, the use of [PCSZ](#) allows for better optimization by the compiler and is more semantically compatible with C++.

The [PCSZ](#) data type is defined in addition to the [PSZ](#) data type. Existing code that calls functions that use [PSZ](#) will continue to work correctly, though they may cause warning or error messages when compiled.

LINK386

The C++ compiler will provide a dynamic link library which is be used by LINK386, or other OS/2 linker, when generating error messages.

This DLL will convert a compiler generated mangled name into the function prototype. If the DLL is not present, an error message will be displayed and LINK386 will display the compiler-generated mangled name in error messages.

Device Functions

This section describes functions that an application would use to query or access devices.

DevCloseDC

DevCloseDC - Syntax

This function closes a device context.

```
#define INCL_DEV /* Or use INCL_PM, Also in COMMON section */
#include <os2.h>

HDC     hdc; /* Device-context handle. */
HMF     hmf; /* Error indicator metafile handle (for a metafile device context) */

hmf = DevCloseDC(hdc);
```

DevCloseDC Parameter - hdc

hdc (**HDC**) - input
Device-context handle.

DevCloseDC Return Value - hmf

hmf (**HMF**) - returns
Error indicator metafile handle (for a metafile device context)

DEV_ERROR	Error occurred.
DEV_OK	Device closed, but not a metafile device context.
Other	Device closed, a metafile device context whose metafile handle is returned.

DevCloseDC - Parameters

hdc ([HDC](#)) - input
Device-context handle.

hmf ([HMF](#)) - returns
Error indicator metafile handle (for a metafile device context)

DEV_ERROR	Error occurred.
DEV_OK	Device closed, but not a metafile device context.
Other	Device closed, a metafile device context whose metafile handle is returned.

DevCloseDC - Remarks

If the device context is currently associated with a presentation space, or if it is created with the [WinOpenWindowDC](#) call (that is, it is a screen device context), an error is raised, and the device context is not closed.

If the device context being closed is a memory device context that has a bit map currently selected into it (see "GpiSetBitmap" in the *Presentation Manager Programming Reference*) the bit map is automatically deselected before the device context is closed.

Any clip region currently in use for this device context is deleted.

DevCloseDC - Errors

Possible returns from [WinGetLastError](#)

PMERR_NOT_CREATED_BY_DEVOPENDC (0x20DC)
An attempt has been made to destroy a device context using DevCloseDC that was not created using [DevOpenDC](#).

PMERR_DC_IS_ASSOCIATED (0x2017)
An attempt was made to associate a presentation space with a device context that was already associated or to destroy a device context that was associated.

PMERR_INV_HDC (0x207C)
An invalid device-context handle or (micro presentation space) presentation-space handle was specified.

DevCloseDC - Related Functions

Prerequisite Functions

- [DevOpenDC](#)

Related Functions

- [WinOpenWindowDC](#)

DevCloseDC - Example Code

This example calls DevCloseDC to close a device context based on the handle returned from DevOpenDC.

```
#define INCL_DEV                      /* Device Function definitions */
#include <os2.h>

HDC hdc;                             /* Device-context handle */
HMF hmf;                             /* error code (or metafile handle if
                                     metafile device context) */

/* close the device context associated with handle hdc */
hmf = DevCloseDC(hdc);
```

DevCloseDC - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DevEscape

DevEscape - Syntax

This function allows applications to access facilities of a device not otherwise available through the API. Escapes are, in general, sent to the presentation driver and must be understood by it.

```
#define INCL_DEV /* Or use INCL_PM, */
#include <os2.h>

HDC     hdc;          /* Device-context handle. */
LONG    lCode;        /* Escape code. */
LONG    lInCount;     /* Input data count. */
PBYTE   pbInData;     /* The input data required for this escape. */
PLONG    plOutCount;  /* Output data count. */
PBYTE   pbOutData;    /* Output data. */
LONG    lResult;      /* Implementation error indicator: */

lResult = DevEscape(hdc, lCode, lInCount,
```

```
pbInData, plOutCount, pbOutData);
```

DevEscape Parameter - hdc

hdc ([HDC](#)) - input
Device-context handle.

DevEscape Parameter - ICode

ICode ([LONG](#)) - input
Escape code.

If the device context is of type OD_QUEUED with a PM_Q_STD spool file, some escapes are sent to the presentation driver and others are recorded in the spool file (depending on the escape code). If the device context is of type OD_METAFILE, all escapes are metafiled. If the device context is of any type other than OD_QUEUED (with a PM_Q_STD spool file) or OD_METAFILE, all escapes are sent to the presentation driver.

The description for each standard escape specifies which of these categories the escape falls into.

Devices can define additional escape functions using user *ICode* values, that have the following ranges:

32 768 through 40 959	Not metafiled and not recorded (sent to presentation driver for PM_Q_STD)
40 960 through 49 151	Metafiled only (sent to presentation driver for PM_Q_STD)
49 152 through 57 343	Metafiled and recorded (not sent to presentation driver) for PM_Q_STD
57 344 through 65 535	Recorded only (not sent to presentation driver for PM_Q_STD).

The following escapes are defined:

DEVESC_QUERYESCSUPPORT
DEVESC_GETSCALINGFACTOR
DEVESC_STARTDOC
DEVESC_ENDDOC
DEVESC_ABORTDOC
DEVESC_NEWFRAME
DEVESC_RAWDATA
DEVESC_QUERYVIOCELLSIZES
DEVESC_SETMODE

DevEscape Parameter - IInCount

lInCount (**LONG**) - input
Input data count.

Number of bytes of data in the *pbInData* buffer.

DevEscape Parameter - pbInData

pbInData (**PBYTE**) - input
The input data required for this escape.

DevEscape Parameter - plOutCount

plOutCount (**PLONG**) - in/out
Output data count.

plOutCount is the number of bytes of data in the *pbOutData* buffer.

If data is returned in *pbOutData* , *plOutCount* is updated to the number of bytes of data returned.

DevEscape Parameter - pbOutData

pbOutData (**PBYTE**) - output
Output data.

pbOutData is a buffer that receives the output from this escape. If *plOutCount* is null, no data is returned.

DevEscape Return Value - IResult

IResult (**LONG**) - returns
Implementation error indicator:

DEVESC_ERROR	Error
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEV_OK	OK.

DevEscape - Parameters

hdc ([HDC](#)) - input
Device-context handle.

ICode ([LONG](#)) - input
Escape code.

If the device context is of type OD_QUEUED with a PM_Q_STD spool file, some escapes are sent to the presentation driver and others are recorded in the spool file (depending on the escape code). If the device context is of type OD_METAFILE, all escapes are metafiled. If the device context is of any type other than OD_QUEUED (with a PM_Q_STD spool file) or OD_METAFILE, all escapes are sent to the presentation driver.

The description for each standard escape specifies which of these categories the escape falls into.

Devices can define additional escape functions using user *ICode* values, that have the following ranges:

32 768 through 40 959	Not metafiled and not recorded (sent to presentation driver for PM_Q_STD)
40 960 through 49 151	Metafiled only (sent to presentation driver for PM_Q_STD)
49 152 through 57 343	Metafiled and recorded (not sent to presentation driver) for PM_Q_STD
57 344 through 65 535	Recorded only (not sent to presentation driver for PM_Q_STD).

The following escapes are defined:

DEVESC_QUERYESCSUPPORT
DEVESC_GETSCALINGFACTOR
DEVESC_STARTDOC
DEVESC_ENDDOC
DEVESC_ABORTDOC
DEVESC_NEWFRAME
DEVESC_RAWDATA
DEVESC_QUERYVIOCELLSIZES
DEVESC_SETMODE

lInCount ([LONG](#)) - input
Input data count.

Number of bytes of data in the *pbInData* buffer.

pbInData ([PBYTE](#)) - input
The input data required for this escape.

pOutCount ([PLONG](#)) - in/out
Output data count.

pOutCount is the number of bytes of data in the *pbOutData* buffer.

If data is returned in *pbOutData*, *pOutCount* is updated to the number of bytes of data returned.

pbOutData ([PBYTE](#)) - output
Output data.

pbOutData is a buffer that receives the output from this escape. If *pOutCount* is null, no data is returned.

IResult (LONG) - returns
Implementation error indicator:

DEVESC_ERROR	Error
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEV_OK	OK.

DevEscape - Remarks

The data fields for standard escapes are:

DEVESC_QUERYESCSUPPORT
Queries whether a particular escape is implemented by the presentation driver. The return value gives the result.

This escape is not metafiled or recorded.

<i>lInCount</i>	Number of bytes pointed to by <i>pbInData</i> .
<i>pbInData</i>	The buffer contains an escape code value specifying the escape function to be checked.
<i>plOutCount</i>	Not used; can be set to 0.
<i>pbOutData</i>	Not used; can be set to null.

DEVESC_GETSCALINGFACTOR
Returns the scaling factors for the x and y axes of a printing device. For each scaling factor, an exponent of two is put in *pbOutData*. Thus, the value 3 is used if the scaling factor is 8.

Scaling factors are used by devices that cannot support graphics at the same resolution as the device resolution.

This escape is not metafiled or recorded.

<i>lInCount</i>	Not used; can be set to 0.
<i>pbInData</i>	Not used; can be set to null.
<i>plOutCount</i>	The number of bytes of data pointed to by <i>pbOutData</i> . On return, this is updated to the number of bytes returned.
<i>pbOutData</i>	The address of a SFACTORS structure, which on return contains the scaling factors for the x and y axes.

DEVESC_STARTDOC
Indicates that a new print job is starting. All subsequent output to the device context is spooled under the same job identifier until a DEVESC_ENDDOC occurs.

GpiAssociate must be issued to associate the presentation space with the device context before issuing this escape.

This escape is metafiled but not recorded.

<i>lInCount</i>	Number of bytes pointed to by <i>pbInData</i> .
<i>pbInData</i>	The buffer contains a null-terminated string, specifying the name of the document.
<i>plOutCount</i>	Not used; can be set to 0.
<i>pbOutData</i>	Not used; can be set to null.

DEVESC_ENDDOC
Ends a print job started by DEVESC_STARTDOC.

This escape is metafiled but not recorded.

<i>lInCount</i>	Not used; can be set to 0.
<i>pbInData</i>	Not used; can be set to null.
<i>plOutCount</i>	Set equal to 2.
<i>pbOutData</i>	The buffer contains a USHORT specifying the job identifier if a spooler print job is created.

DEVESC_ABORTDOC
Aborts the current job, erasing everything the application has written to the device since the last DEVESC_STARTDOC, including the DEVESC_STARTDOC.

This escape is metafiled but not recorded.

<i>lInCount</i>	Not used; can be set to 0
<i>pbInData</i>	Not used; can be set to null
<i>pOutCount</i>	Not used; can be set to 0
<i>pbOutData</i>	Not used; can be set to null.

DEVESC_NEWFRAME

Signals when an application has finished writing to a page and wants to start a new page. It is similar to GpiErase processing for a screen device context, and causes a reset of the attributes. This escape is used with a printer device to advance to a new page.

This escape is metafiled and recorded.

<i>lInCount</i>	Not used; can be set to 0
<i>pbInData</i>	Not used; can be set to null
<i>pOutCount</i>	Not used; can be set to 0
<i>pbOutData</i>	Not used; can be set to null.

DEVESC_RAWDATA

Allows an application to send data directly to a presentation driver. For example, in the case of a printer driver, this could be a printer data stream.

If DEVESC_RAWDATA is mixed with other data (such as GPI data) being sent to the same page of a device context, the results are unpredictable and depend upon the action taken by the presentation driver. For example, a presentation driver might ignore GPI data if DEVESC_RAWDATA is mixed with it on the same page. In general, DEVESC_RAWDATA should be sent either to a separate page (using the DEVESC_NEWFRAME escape to obtain a new page) or to a separate document (using the DEVESC_STARTDOC and DEVESC_ENDDOC escapes to create a new document).

This escape is metafiled and recorded.

<i>lInCount</i>	Number of bytes pointed to by <i>pbInData</i>
<i>pbInData</i>	Pointer to the raw data
<i>pOutCount</i>	Not used; can be set to 0
<i>pbOutData</i>	Not used; can be set to null.

DEVESC_QUERYVIOCELLSIZES

Returns the VIO cell sizes supported by the presentation driver.

This escape is not metafiled or recorded.

<i>lInCount</i>	Not used; can be set to 0
<i>pbInData</i>	Not used; can be set to null.
<i>pOutCount</i>	The number of bytes of data pointed to by <i>pbOutData</i> . It must be an even multiple of the size in bytes of the LONG data type. On return, this is updated to the number of bytes returned.
<i>pbOutData</i>	<p>The address of a buffer, which on return contains a VIO_SIZE_COUNT structure, immediately followed by <i>count</i> copies of a VIO_FONT_CELL_SIZE structure.</p> <p>If <i>pOutCount</i> is less than the size of a LONG data type, <i>pOutCount</i> is updated to zero, and nothing is returned in the buffer pointed to by <i>pbOutData</i>.</p> <p>If <i>pOutCount</i> is equal to the size of a LONG data type, <i>pbOutData</i> returns the number of VIO cell sizes that can be returned by this escape. The buffer pointed to by <i>pbOutData</i> is updated so that <i>maxcount</i> is the number of VIO cell sizes that can be returned.</p> <p>If <i>pOutCount</i> is greater than the size of a LONG data type, <i>pbOutData</i> returns the VIO cell sizes that are supported. The buffer pointed to by <i>pbOutData</i> is updated so that:</p> <ul style="list-style-type: none">• <i>maxcount</i> is the number of VIO cell sizes that can be returned• <i>count</i> is the number of VIO cell sizes returned (may be zero if <i>pOutCount</i> is equal to twice the size of a LONG data type)• <i>count</i> copies of a VIO_FONT_CELL_SIZE structure are returned.

DEVESC_SETMODE

Sets the printer into a particular mode. It is optional for printer drivers to support this escape, but those that do support it need to be aware of the code page of any built-in fonts. For example, if only code page 437 is built in, it is used if 437 is requested by DEVESC_SETMODE; however, if code page 865 is requested, a suitable code page/font could be downloaded.

This escape is metafiled and recorded.

lInCount
pbInData
plOutCount
pbOutData

Number of bytes pointed to by *pbInData*
Buffer contains an [ESCSETMODE](#) structure
Not used; can be set to 0
Not used; can be set to null.

DevEscape - Errors

Possible returns from [WinGetLastError](#)

PMERR_INV_ESC_CODE (0x206D)

An invalid escape code was used in a call to DevEscape.

PMERR_INV_HDC (0x207C)

An invalid device-context handle or (micro presentation space) presentation-space handle was specified.

PMERR_INV_LENGTH_OR_COUNT (0x2092)

An invalid length or count parameter was specified.

PMERR_ESC_CODE_NOT_SUPPORTED (0x202B)

The code specified with DevEscape is not supported by the target device driver.

PMERR_INV_ESCAPE_DATA (0x206E)

An invalid data parameter was specified with DevEscape.

DevEscape - Related Functions

Prerequisite Functions

- [DevOpenDC](#)

Related Functions

- GpiAssociate (for DEVESC_STARTDOC)
- GpiErase (for DEVESC_NEWFRAME)

DevEscape - Graphic Elements and Orders

DevEscape functions generate orders only when metafileing.

Order: GEESCP

DevEscape - Example Code

This example uses DevEscape to access facilities of a device that would otherwise be unavailable through the normal Device API set. Here,

a new page in a print job is started.

```
#define INCL_DEV          /* Device Function definitions */
#include <os2.h>

LONG  lResult;           /* Error code or not implemented
                           warning code */
HDC   hdc;               /* Device-context handle */
LONG  plOutCount;        /* length of output buffer(input),
                           number of bytes returned(output) */
PBYTE pbOutData;         /* output buffer */

/* for the NEWFRAME, input and output buffers are not used,
   so set the buffer lengths to zero(0) and set the buffers to
   NULL */
plOutCount = 0;
pbOutData = NULL;

lResult = DevEscape(hdc, DEVEscape_NEWFRAME, 0L, NULL, &plOutCount,
                    pbOutData);
```

DevEscape - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Graphic Elements and Orders](#)
[Glossary](#)

DevOpenDC

DevOpenDC - Syntax

This function creates a device context.

```
#define INCL_DEV /* Or use INCL_PM, Also in COMMON section */
#include <os2.h>

HAB      hab;           /* Anchor-block handle. */
LONG     lType;          /* Type of device context: */
PSZ      pszToken;       /* Device-information token. */
LONG     lCount;         /* Number of items. */
PDEVOPENDATA pdopData;   /* Open-device-context data area. */
HDC      hdcComp;        /* Compatible-device-context handle. */
HDC      hdc;            /* Device-context handle: */

hdc = DevOpenDC(hab, lType, pszToken, lCount,
                pdopData, hdcComp);
```

DevOpenDC Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

DevOpenDC Parameter - IType

IType ([LONG](#)) - input
Type of device context:

- OD_QUEUED**
A device, such as a printer or plotter, for which output is to be queued.
Certain restrictions apply for this device type;
- OD_DIRECT**
A device, such as a printer or plotter, for which output is not to be queued.
- OD_INFO**
A device, such as a printer or plotter, but the device context is used only to retrieve information (for example, font metrics). Drawing can be performed to a presentation space associated with such a device context, but no output medium is updated.
- OD_METAFILE**
The device context is used to write a metafile. The presentation page defines the area of interest within the picture in the metafile. See [OD_METAFILE_NOQUERY](#).
Certain restrictions apply for this device type;
- OD_METAFILE_NOQUERY**
The device context is used to write a metafile.
Functionally, this device type is the same as [OD_METAFILE](#), except that querying of attributes is not allowed with a presentation space while it is associated with an [OD_METAFILE_NOQUERY](#) device context. If querying of attributes is not required, [OD_METAFILE_NOQUERY](#) should be used in preference to [OD_METAFILE](#), since it gives improved performance.
Certain restrictions apply for this device type;
- OD_MEMORY**
A device context that is used to contain a bit map. The *hdcComp* parameter identifies a device with which the memory device context is to be compatible.
-

DevOpenDC Parameter - pszToken

pszToken ([PSZ](#)) - input
Device-information token.

This identifies the device information, held in the initialization file. This information is the same as that which may be pointed to by

pdopData; any information that is obtained from *pdopData* overrides the information obtained by using this parameter.

If *pszToken* is specified as "", no device information is taken from the initialization file.

OS/2 behaves as if "" is specified, but it allows any string.

DevOpenDC Parameter - ICount

ICount (**LONG**) - input
Number of items.

This is the number of items present in the *pdopData* parameter. This can be less than the full list if omitted items are irrelevant, or are supplied from *pszToken* or elsewhere.

DevOpenDC Parameter - pdopData

pdopData (**PDEVOPENDATA**) - input
Open-device-context data area.

DevOpenDC Parameter - hdcComp

hdcComp (**HDC**) - input
Compatible-device-context handle.

When */Type* is OD_MEMORY, this parameter is a handle to a device context compatible with bit maps that are to be used with this device context.

If *hdcComp* is NULLHANDLE, compatibility with the screen is assumed.

DevOpenDC Return Value - hdc

hdc (**HDC**) - returns
Device-context handle:

DEV_ERROR

Error

<>0

Device-context handle.

DevOpenDC - Parameters

hab (**HAB**) - input
Anchor-block handle.

IType (**LONG**) - input
Type of device context:

- OD_QUEUED**
A device, such as a printer or plotter, for which output is to be queued.
Certain restrictions apply for this device type;
- OD_DIRECT**
A device, such as a printer or plotter, for which output is not to be queued.
- OD_INFO**
A device, such as a printer or plotter, but the device context is used only to retrieve information (for example, font metrics). Drawing can be performed to a presentation space associated with such a device context, but no output medium is updated.
- OD_METAFILE**
The device context is used to write a metafile. The presentation page defines the area of interest within the picture in the metafile. See **OD_METAFILE_NOQUERY**.
Certain restrictions apply for this device type;
- OD_METAFILE_NOQUERY**
The device context is used to write a metafile.
Functionally, this device type is the same as **OD_METAFILE**, except that querying of attributes is not allowed with a presentation space while it is associated with an **OD_METAFILE_NOQUERY** device context. If querying of attributes is not required, **OD_METAFILE_NOQUERY** should be used in preference to **OD_METAFILE**, since it gives improved performance.
Certain restrictions apply for this device type;
- OD_MEMORY**
A device context that is used to contain a bit map. The *hdcComp* parameter identifies a device with which the memory device context is to be compatible.

pszToken (**PSZ**) - input
Device-information token.

This identifies the device information, held in the initialization file. This information is the same as that which may be pointed to by *pdopData*; any information that is obtained from *pdopData* overrides the information obtained by using this parameter.

If *pszToken* is specified as **""**, no device information is taken from the initialization file.

OS/2 behaves as if **""** is specified, but it allows any string.

ICount (**LONG**) - input
Number of items.

This is the number of items present in the *pdopData* parameter. This can be less than the full list if omitted items are irrelevant, or are supplied from *pszToken* or elsewhere.

pdopData (**PDEVOPENDATA**) - input
Open-device-context data area.

hdcComp (**HDC**) - input
Compatible-device-context handle.

When *IType* is **OD_MEMORY**, this parameter is a handle to a device context compatible with bit maps that are to be used with this device context.

If *hdcComp* is **NULLHANDLE**, compatibility with the screen is assumed.

hdc (**HDC**) - returns
Device-context handle:

DEV_ERROR

<>0 Error
Device-context handle.

DevOpenDC - Remarks

A device context is a means of writing to a particular device. Before using GPI functions to cause output to be directed to the device context, the GpiAssociate function call must be issued (or the GPIA_ASSOC option specified on GpiCreatePS).

DevOpenDC cannot be used to open a device context for a screen window; use [WinOpenWindowDC](#) instead.

The device context is owned by the process from which DevOpenDC is issued. It cannot be accessed directly from any other process. If it still exists when the process terminates, it is automatically deleted by the system. When using a device context type of OD_METAFILE_NOQUERY the querying of attributes is not allowed. To improve performance of this type of metafile no error checking is performed to ensure that such API calls are not attempted. Query calls are accepted but the results returned are undefined.

This function requires the existence of a message queue.

DevOpenDC - Errors

Possible returns from [WinGetLastError](#)

PMERR_INV_DC_TYPE (0x2060)

An invalid type parameter was specified with DevOpenDC, or a function was issued that is invalid for a OD_METAFILE_NOQUERY device context.

PMERR_INV_LENGTH_OR_COUNT (0x2092)

An invalid length or count parameter was specified.

PMERR_INV_DC_DATA (0x205F)

An invalid data parameter was specified with DevOpenDC.

PMERR_INV_HDC (0x207C)

An invalid device-context handle or (micro presentation space) presentation-space handle was specified.

PMERR_INV_DRIVER_NAME (0x2067)

A driver name was specified which has not been installed.

PMERR_INV_LOGICAL_ADDRESS (0x2097)

An invalid device logical address was specified.

DevOpenDC - Related Functions

Prerequisite Functions

- [WinInitialize](#)

Related Functions

- [DevCloseDC](#)
- [PrfQueryProfileString](#)
- [WinOpenWindowDC](#)
- [WinQueryWindow](#)

DevOpenDC - Example Code

This example calls DevOpenDC to create a memory device context with screen compatibility and then associates that context with a newly created presentation space.

```
#define INCL_DEV                /* Device Function definitions */
#define INCL_GPICONTROL        /* GPI control Functions */
#include <os2.h>

HDC   hdc;                /* Device-context handle */
HAB   hab;                /* Anchor-block handle */
/* context data structure */
DEVOPENSTRUC dop = {NULL, "DISPLAY", NULL, NULL, NULL, NULL,
                    NULL, NULL, NULL};
HPS   hps;                /* presentation-space handle */
SIZEL sizl={0, 0};        /* use same page size as device */

/* create memory device context */
hdc = DevOpenDC(hab, OD_MEMORY, "", 5L, (PDEVOPENDATA)&dop, NULLHANDLE);

/* create a presentation space associated with the context */
hps = GpiCreatePS(hab, hdc, &sizl, GPIA_ASSOC | PU_PELS);
```

DevOpenDC - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DevPostDeviceModes

DevPostDeviceModes - Syntax

This function returns, and optionally sets job properties.

```
#define INCL_DEV /* Or use INCL_PM, */
#include <os2.h>

HAB   hab;                /* Anchor-block handle. */
PDRIVDATA pdrvDriverData; /* Driver data. */
PSZ    pszDriverName;     /* Device-driver name. A string containing the name of the presentation driver;
```

```

PSZ          pszDeviceName;    /* Device-type name. */
PSZ          pszName;          /* Device name. */
ULONG        flOptions;        /* Dialog options. */
LONG         lDriverCount;     /* Size/error indicator. */

lDriverCount = DevPostDeviceModes(hab, pdrvDriverData,
                                   pszDriverName, pszDeviceName,
                                   pszName, flOptions);

```

DevPostDeviceModes Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

DevPostDeviceModes Parameter - pdrvDriverData

pdrvDriverData ([PDRIVDATA](#)) - in/out
Driver data.

A data area that, on return, contains device data defined by the presentation driver. If the pointer to the area is NULL, this function returns the required size of the data area.

The format of the data is the same as that which occurs within the [DEVOPENSTRUC](#) structure, passed on the *pdrvData* parameter of [DevOpenDC](#).

DevPostDeviceModes Parameter - pszDriverName

pszDriverName ([PSZ](#)) - input
Device-driver name. A string containing the name of the presentation driver; for example, "LASERJET".

DevPostDeviceModes Parameter - pszDeviceName

pszDeviceName ([PSZ](#)) - input
Device-type name.

Null-terminated string in a 32-byte field, identifying the device type; for example, "HP LaserJet IID" (model number). Valid names are defined by device drivers.

Note: This parameter always overrides the data in the *szDeviceName* field of the [DRIVDATA](#) structure, passed in the *pdrvDriverData* parameter.

DevPostDeviceModes Parameter - pszName

pszName ([PSZ](#)) - input
Device name.

A name that identifies the device; for example, "PRINTER1".

DevPostDeviceModes Parameter - flOptions

flOptions ([ULONG](#)) - input
Dialog options.

Options that control whether a dialog is displayed.

DPDM_POSTJOBPROP

This function allows the user to set properties for the print job by displaying a dialog and returning the updated job properties. Examples of job properties are paper size, paper orientation, and single-sided or duplex.

The printer is configured in the shell using a dialog provided by the presentation driver. The configuration describes the actual printer setup such as number of paper bins, available paper sizes, and any installed hardware fonts.

Before the job properties dialog is displayed the presentation driver merges any changes in the printer configuration with the data passed in the *pdrvDriverData* parameter. This allows, for example new paper sizes to be added into the job properties dialog. The parameter *pszName* can be specified as NULL although this is not recommended because the presentation driver cannot easily find the printer configuration to merge.

It is the responsibility of the application to retrieve and store job properties. An application can choose to store job properties either on a per document or per application basis. The job properties can then be passed into [DevOpenDC](#). Initial (default) job properties can be retrieved using DPDM_QUERYJOBPROP option.

The application cannot tell if the user modified the job properties or just cancelled the dialog. Hence the job properties returned in the *pdrvDriverData* parameter must always be stored.

The shell allows users to specify default job properties for a printer. The spooler API [SplQueryQueue](#) can be used to retrieve these defaults. The spooler automatically adds the default job properties for a printer to any jobs that are submitted without job properties.

DPDM_QUERYJOBPROP

Do not display a dialog. Return the default job properties. These defaults are derived from the defaults for the chosen device; for example, "HP Laserjet IID" and the printer setup specified via the shell printer driver configuration dialog.

DevPostDeviceModes Return Value - IDriverCount

IDriverCount ([LONG](#)) - returns
Size/error indicator.

Value depends on what was passed as the pointer to *pdrvDriverData*. If NULL was passed, the following values are possible:

DPDM_ERROR
DPDM_NONE
>0

Error
No settable options
Size in bytes required for *pdrvDriverData*.

If any other value was passed, the following values are possible:

DPDM_ERROR	Error
DPDM_NONE	No settable options
DEV_OK	OK.

DevPostDeviceModes - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

pdrvDriverData ([PDRIVDATA](#)) - in/out
Driver data.

A data area that, on return, contains device data defined by the presentation driver. If the pointer to the area is NULL, this function returns the required size of the data area.

The format of the data is the same as that which occurs within the [DEVOPENSTRUC](#) structure, passed on the *pdrvData* parameter of [DevOpenDC](#).

pszDriverName ([PSZ](#)) - input
Device-driver name. A string containing the name of the presentation driver; for example, "LASERJET".

pszDeviceName ([PSZ](#)) - input
Device-type name.

Null-terminated string in a 32-byte field, identifying the device type; for example, "HP LaserJet IID" (model number). Valid names are defined by device drivers.

Note: This parameter always overrides the data in the *szDeviceName* field of the [DRIVDATA](#) structure, passed in the *pdrvDriverData* parameter.

pszName ([PSZ](#)) - input
Device name.

A name that identifies the device; for example, "PRINTER1".

flOptions ([ULONG](#)) - input
Dialog options.

Options that control whether a dialog is displayed.

DPDM_POSTJOBPROP

This function allows the user to set properties for the print job by displaying a dialog and returning the updated job properties. Examples of job properties are paper size, paper orientation, and single-sided or duplex.

The printer is configured in the shell using a dialog provided by the presentation driver. The configuration describes the actual printer setup such as number of paper bins, available paper sizes, and any installed hardware fonts.

Before the job properties dialog is displayed the presentation driver merges any changes in the printer configuration with the data passed in the *pdrvDriverData* parameter. This allows, for example new paper sizes to be added into the job properties dialog. The parameter *pszName* can be specified as NULL although this is not recommended because the presentation driver cannot easily find the printer configuration to merge.

It is the responsibility of the application to retrieve and store job properties. An application can choose to store job properties either on a per document or per application basis. The job properties can then be passed into [DevOpenDC](#). Initial (default) job properties can be retrieved using DPDM_QUERYJOBPROP option.

The application cannot tell if the user modified the job properties or just cancelled the dialog. Hence the job properties returned in the *pdrvDriverData* parameter must always be stored.

The shell allows users to specify default job properties for a printer. The spooler API [SpiQueryQueue](#) can be used to retrieve these defaults. The spooler automatically adds the default job properties for a printer to any jobs that are submitted without job properties.

DPDM_QUERYJOBPROP

Do not display a dialog. Return the default job properties. These defaults are derived from the defaults for the chosen device; for example, "HP Laserjet IID" and the printer setup specified via the shell printer driver configuration dialog.

IDriverCount ([LONG](#)) - returns
Size/error indicator.

Value depends on what was passed as the pointer to *pdrivDriverData*. If NULL was passed, the following values are possible:

DPDM_ERROR	Error
DPDM_NONE	No settable options
>0	Size in bytes required for <i>pdrivDriverData</i> .

If any other value was passed, the following values are possible:

DPDM_ERROR	Error
DPDM_NONE	No settable options
DEV_OK	OK.

DevPostDeviceModes - Remarks

An application can first call this function with a NULL data pointer to find out how much storage is needed for the data area. Having allocated the storage, the application can then make the call a second time for the data to be entered. The returned data can then be passed in [DevOpenDC](#) as *pdrivDriverData* within the *pdpData* parameter.

Calling this function requires the existence of a message queue.

Use [SplEnumDevice](#) or [SplEnumPrinter](#) with *//Type* set to SPL_PR_DIRECT_DEVICE or SPL_PR_QUEUED_DEVICE to get a list of all the devices.

To get information about a specific device use [SplQueryDevice](#).

DevPostDeviceModes - Errors

Possible returns from [WinGetLastError](#)

PMERR_INV_DRIVER_DATA (0x2066)
Invalid driver data was specified.

PMERR_DRIVER_NOT_FOUND (0x2026)
The device driver specified with DevPostDeviceModes was not found.

PMERR_INV_DEVICE_NAME (0x2061)
An invalid devicename parameter was specified with DevPostDeviceModes.

PMERR_INV_LOGICAL_ADDRESS (0x2097)
An invalid device logical address was specified.

DevPostDeviceModes - Related Functions

Related Functions

- [DevOpenDC](#)

DevPostDeviceModes - Example Code

This example shows how to call DevPostDeviceModes and allocate a new buffer, if necessary, for the larger job properties (DRIVDATA structure).

```
#define INCL_DEV
#define INCL_DOS
#include <os2.h>
#include <memory.h>

{
    ULONG          devrc=FALSE;
    HAB            hab;
    PSZ            pszPrinter;
    HDC            hdc=NULL;
    PDRIVDATA      pOldDrvData;
    PDRIVDATA      pNewDrvData=NULL;
    PDEVOPENSTRUC dops;
    LONG           buflen;

    /* check size of buffer required for job properties */
    buflen = DevPostDeviceModes( hab,
                                NULL,
                                dops->pszDriverName,
                                dops->pdriv->szDeviceName,
                                pszPrinter,
                                DPDM_POSTJOBPROP
                                );

    /* return error to caller */
    if (buflen<=0)
        return(buflen);

    /* allocate some memory for larger job properties and */
    /* return error to caller */

    if (buflen != dops->pdriv->cb)
    {
        if (DosAllocMem((PVOID)&pNewDrvData,buflen,FALLOC))
            return(DPDM_ERROR);
    }

    /* copy over old data so driver can use old job */
    /* properties as base for job properties dialog */
    pOldDrvData = dops->pdriv;
    dops->pdriv = pNewDrvData;
    memcpy( (PSZ)pNewDrvData, (PSZ)pOldDrvData, pOldDrvData->cb );

    /* display job properties dialog and get updated */
    /* job properties from driver */

    devrc = DevPostDeviceModes( hab,
                                dops->pdriv,
                                dops->pszDriverName,
                                dops->pdriv->szDeviceName,
                                pszPrinter,
                                DPDM_POSTJOBPROP
                                );

    return(devrc);
}
```

DevPostDeviceModes - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DevQueryCaps

DevQueryCaps - Syntax

This function queries the device characteristics.

```
#define INCL_DEV /* Or use INCL_PM, Also in COMMON section */
#include <os2.h>

HDC      hdc;          /* Device-context handle. */
LONG     lStart;        /* First item of information. */
LONG     lCount;        /* Count of items of information. */
PLONG    alArray;       /* Device capabilities. */
BOOL     rc;            /* Success indicator. */

rc = DevQueryCaps(hdc, lStart, lCount, alArray);
```

DevQueryCaps Parameter - hdc

hdc ([HDC](#)) - input
Device-context handle.

DevQueryCaps Parameter - lStart

lStart ([LONG](#)) - input
First item of information.

The number of the first item of information to be returned in *alArray*, counting from zero.

DevQueryCaps Parameter - lCount

ICount (**LONG**) - input
Count of items of information.

This is the count to be returned in *alArray*. It must be greater than zero.

DevQueryCaps Parameter - alArray

alArray (**PLONG**) - output
Device capabilities.

Array of *ICount* elements, starting with *IStart*. The array elements are numbered consecutively, starting with CAPS_FAMILY. The element number constants start with 0.

If *IStart* + *ICount* - 1 exceeds the current highest-defined element number, elements beyond the highest are returned as 0.

CAPS_FAMILY
Device type (values as for *IType* in [DevOpenDC](#)).

CAPS_IO_CAPS
Device input/output capability:

CAPS_IO_DUMMY	Dummy device
CAPS_SUPPORTS_OP	Device supports output
CAPS_SUPPORTS_IP	Device supports input
CAPS_SUPPORTS_IO	Device supports output and input.

CAPS_TECHNOLOGY
Technology:

CAPS_TECH_UNKNOWN	Unknown
CAPS_TECH_VECTOR_PLOTTER	Vector plotter
CAPS_TECH_RASTER_DISPLAY	Raster display
CAPS_TECH_RASTER_PRINTER	Raster printer
CAPS_TECH_RASTER_CAMERA	Raster camera
CAPS_TECH_POSTSCRIPT	PostScript** device.

CAPS_DRIVER_VERSION
Version identifier of the presentation driver.

The high order word of the version identifier is 0. The low order word identifies the release, for example 0x0120 is release 1.2.

CAPS_WIDTH
Media width (for a full screen, maximized window for displays) in pels.

CAPS_HEIGHT
Media depth (for a full screen, maximized window for displays) in pels. (For a plotter, a pel is defined as the smallest possible displacement of the pen and can be smaller than a pen width.)

CAPS_WIDTH_IN_CHARS
Media width (for a full screen, maximized window for displays) in default character columns.

CAPS_HEIGHT_IN_CHARS
Media depth (for a full screen, maximized window for displays) in default character rows.

CAPS_HORIZONTAL_RESOLUTION
Horizontal resolution of device in pels per meter.

CAPS_VERTICAL_RESOLUTION
Vertical resolution of device in pels per meter.

CAPS_CHAR_WIDTH

	Default character-box width in pels for VIO.	
CAPS_CHAR_HEIGHT	Default character-box height in pels for VIO.	
CAPS_SMALL_CHAR_WIDTH	Default small-character box width in pels for VIO. This is 0 if there is only one character-box size.	
CAPS_SMALL_CHAR_HEIGHT	Default small-character box height in pels for VIO. This is 0 if there is only one character-box size.	
CAPS_COLORS	Number of distinct colors supported at the same time, including reset (gray scales count as distinct colors). If loadable color tables are supported, this is the number of entries in the device color table. For plotters, the value returned is the number of pens plus one (for the background).	
CAPS_COLOR_PLANES	Number of color planes.	
CAPS_COLOR_BITCOUNT	Number of adjacent color bits for each pel (within one plane).	
CAPS_COLOR_TABLE_SUPPORT	Loadable color table support:	
	CAPS_COLTABL_RGB_8	1 if RGB color table can be loaded, with a minimum support of 8 bits each for red, green, and blue.
	CAPS_COLTABL_RGB_8_PLUS	1 if color table with other than 8 bits for each primary color can be loaded.
	CAPS_COLTABL_TRUE_MIX	1 if true mixing occurs when the logical color table has been realized, providing that the size of the logical color table is not greater than the number of distinct colors supported (see element CAPS_COLORS).
	CAPS_COLTABL_REALIZE	1 if a loaded color table can be realized.
CAPS_MOUSE_BUTTONS	The number of pointing device buttons that are available. A returned value of 0 indicates that there are no pointing device buttons available.	
CAPS_FOREGROUND_MIX_SUPPORT	Foreground mix support:	
	CAPS_FM_OR	Logical OR.
	CAPS_FM_OVERPAINT	Overpaint.
	CAPS_FM_XOR	Logical XOR.
	CAPS_FM_LEAVEALONE	Leave alone.
	CAPS_FM_AND	Logical AND.
	CAPS_FM_GENERAL_BOOLEAN	All other mix modes; see "GpiSetMix" in <i>Graphics Programming Interface Programming Reference</i> .
	The value returned is the sum of the values appropriate to the mixes supported. A device capable of supporting OR must, as a minimum, return CAPS_FM_OR + CAPS_FM_OVERPAINT + CAPS_FM_LEAVEALONE, signifying support for the mandatory mixes OR, overpaint, and leave-alone.	
	Note that these numbers correspond to the decimal representation of a bit string that is six bits long, with each bit set to 1 if the appropriate mode is supported.	

Those mixes returned as supported are guaranteed for all primitive types. For more information, see "GpiSetMix" in *Graphics Programming Interface Programming Reference*.

CAPS_BACKGROUND_MIX_SUPPORT

Background mix support:

CAPS_BM_OR	Logical OR.
CAPS_BM_OVERPAINT	Overpaint.
CAPS_BM_XOR	Logical XOR.
CAPS_BM_LEAVEALONE	Leave alone.
CAPS_BM_AND	Logical AND.
CAPS_BM_GENERAL_BOOLEAN	All other mix modes; see "GpiSetMix" in <i>Graphics Programming Interface Programming Reference</i> .
CAPS_BM_SRCTRANSSPARENT	Provides a transparent overlay function by not copying pels from the source bit map to the output bit map if they match the presentation space background color.
CAPS_BM_DESTTRANSPARENT	Provides a transparent underlay function by copying only the pels that match the presentation space background color from the source bit map to the output bit map.

The value returned is the sum of the values appropriate to the mixes supported. A device must, as a minimum, return CAPS_BM_OVERPAINT + CAPS_BM_LEAVEALONE, signifying support for the mandatory background mixes overpaint, and leave-alone.

Note that these numbers correspond to the decimal representation of a bit string that is four bits long, with each bit set to 1 if the appropriate mode is supported.

Those mixes returned as supported are guaranteed for all primitive types. For more information, see "GpiSetMix" in *Graphics Programming Interface Programming Reference*.

CAPS_VIO_LOADABLE_FONTS

Number of fonts that can be loaded for VIO.

CAPS_WINDOW_BYTE_ALIGNMENT

Whether or not the client area of VIO windows should be byte-aligned:

CAPS_BYTE_ALIGN_REQUIRED	Must be byte-aligned.
CAPS_BYTE_ALIGN_RECOMMENDED	More efficient if byte-aligned, but not required.
CAPS_BYTE_ALIGN_NOT_REQUIRED	Does not matter whether byte-aligned.

CAPS_BITMAP_FORMATS

Number of bit-map formats supported by device.

CAPS_RASTER_CAPS

Capability for device raster operations:

CAPS_RASTER_BITBLT	1 if GpiBitBlt and GpiWCBitBlt is supported.
CAPS_RASTER_BANDING	1 if banding is supported
CAPS_RASTER_BITBLT_SCALING	1 if GpiBitBlt and GpiWCBitBlt with scaling is supported.
CAPS_RASTER_SET_PEL	1 if GpiSetPel is supported.
CAPS_RASTER_FONTS	1 if this device can draw raster fonts.
CAPS_RASTER_FLOOD_FILL	1 if GpiFloodFill is supported.
CAPS_MARKER_HEIGHT Default marker-box height in pels.	
CAPS_MARKER_WIDTH Default marker-box width in pels.	
CAPS_DEVICE_FONTS Number of device-specific fonts.	
CAPS_GRAPHICS_SUBSET Graphics drawing subset supported. (3 indicates GOCA DR/3)	
CAPS_GRAPHICS_VERSION Graphics architecture version number supported. (1 indicates Version 1)	
CAPS_GRAPHICS_VECTOR_SUBSET Graphics vector drawing subset supported. (2 indicates GOCA VS/2)	
CAPS_DEVICE_WINDOWING Device windowing support:	
CAPS_DEV_WINDOWING_SUPPORT	1 if device supports windowing.
	Other bits are reserved 0.
CAPS_ADDITIONAL_GRAPHICS Additional graphics support:	
CAPS_GRAPHICS_KERNING_SUPPORT	1 if device supports kerning.
CAPS_FONT_OUTLINE_DEFAULT	1 if device has a default outline font.
CAPS_FONT_IMAGE_DEFAULT	1 if device has a default image

	font.
CAPS_SCALED_DEFAULT_MARKERS	1 if default markers are to be scaled by the marker-box attribute.
CAPS_COLOR_CURSOR_SUPPORT	1 if device supports colored cursors.
CAPS_PALETTE_MANAGER	1 if device supports palette functions (see "GpiCreatePalette" in the <i>Graphics Programming Interface Programming Reference</i>).
CAPS_COSMETIC_WIDELINE_SUPPORT	1 if device supports cosmetic thick lines (see "GpiSetLineWidth" in the <i>Graphics Programming Interface Programming Reference</i>).
	Other bits are reserved 0.
CAPS_PHYS_COLORS	Maximum number of distinct colors available on the device.
CAPS_COLOR_INDEX	Maximum logical color-table index supported for this device. For the EGA and VGA drivers, the value is 63.
CAPS_GRAPHICS_CHAR_WIDTH	Default graphics character-box width, in pels.
CAPS_GRAPHICS_CHAR_HEIGHT	Default graphics character-box height, in pels.

CAPS_HORIZONTAL_FONT_RES

Effective horizontal device resolution in pels per inch, for the purpose of selecting fonts.

For printers, this is the actual device resolution, but for displays it may differ from the actual resolution for reasons of legibility.

CAPS_VERTICAL_FONT_RES

Effective vertical device resolution in pels per inch, for the purpose of selecting fonts.

CAPS_DEVICE_FONT_SIM

Identifies which simulations are valid on device fonts.

Valid flags are:

CAPS_DEV_FONT_SIM_BOLD
CAPS_DEV_FONT_SIM_ITALIC
CAPS_DEV_FONT_SIM_UNDERSCORE
CAPS_DEV_FONT_SIM_STRIKEOUT

CAPS_LINEWIDTH_THICK

Cosmetic thickness of lines and arcs on this device, when *fxLineWidth* is LINEWIDTH_THICK (see "GpiSetLineWidth" in the *Graphics Programming Interface Programming Reference*). The units are pels. A value of 0 is interpreted as 2 pels.

CAPS_DEVICE_POLYSET_POINTS

Number of points in a polyset that a device can handle.

DevQueryCaps Return Value - rc

rc (**BOOL**) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

DevQueryCaps - Parameters

hdc (**HDC**) - input

Device-context handle.

IStart (**LONG**) - input

First item of information.

The number of the first item of information to be returned in *alArray*, counting from zero.

ICount (**LONG**) - input

Count of items of information.

This is the count to be returned in *alArray*. It must be greater than zero.

alArray (**PLONG**) - output

Device capabilities.

Array of *ICount* elements, starting with *IStart*. The array elements are numbered consecutively, starting with CAPS_FAMILY. The element number constants start with 0.

If */Start + /Count - 1* exceeds the current highest-defined element number, elements beyond the highest are returned as 0.

CAPS_FAMILY

Device type (values as for */Type* in [DevOpenDC](#)).

CAPS_IO_CAPS

Device input/output capability:

CAPS_IO_DUMMY	Dummy device
CAPS_SUPPORTS_OP	Device supports output
CAPS_SUPPORTS_IP	Device supports input
CAPS_SUPPORTS_IO	Device supports output and input.

CAPS_TECHNOLOGY

Technology:

CAPS_Tech_UNKNOWN	Unknown
CAPS_Tech_VECTOR_PLOTTER	Vector plotter
CAPS_Tech_RASTER_DISPLAY	Raster display
CAPS_Tech_RASTER_PRINTER	Raster printer
CAPS_Tech_RASTER_CAMERA	Raster camera
CAPS_Tech_POSTSCRIPT	PostScript** device.

CAPS_DRIVER_VERSION

Version identifier of the presentation driver.

The high order word of the version identifier is 0. The low order word identifies the release, for example 0x0120 is release 1.2.

CAPS_WIDTH

Media width (for a full screen, maximized window for displays) in pels.

CAPS_HEIGHT

Media depth (for a full screen, maximized window for displays) in pels. (For a plotter, a pel is defined as the smallest possible displacement of the pen and can be smaller than a pen width.)

CAPS_WIDTH_IN_CHARS

Media width (for a full screen, maximized window for displays) in default character columns.

CAPS_HEIGHT_IN_CHARS

Media depth (for a full screen, maximized window for displays) in default character rows.

CAPS_HORIZONTAL_RESOLUTION

Horizontal resolution of device in pels per meter.

CAPS_VERTICAL_RESOLUTION

Vertical resolution of device in pels per meter.

CAPS_CHAR_WIDTH

Default character-box width in pels for VIO.

CAPS_CHAR_HEIGHT

Default character-box height in pels for VIO.

CAPS_SMALL_CHAR_WIDTH

Default small-character box width in pels for VIO. This is 0 if there is only one character-box size.

CAPS_SMALL_CHAR_HEIGHT

Default small-character box height in pels for VIO. This is 0 if there is only one character-box size.

CAPS_COLORS

Number of distinct colors supported at the same time, including reset (gray scales count as distinct colors). If loadable color tables are supported, this is the number of entries in the device color table. For plotters, the value returned is the number of pens plus one (for the background).

CAPS_COLOR_PLANES

Number of color planes.

CAPS_COLOR_BITCOUNT

Number of adjacent color bits for each pel (within one plane).

CAPS_COLOR_TABLE_SUPPORT

Loadable color table support:

CAPS_COLTABL_RGB_8	1 if RGB color table can be loaded, with a minimum support of 8 bits each for red, green, and blue.
CAPS_COLTABL_RGB_8_PLUS	1 if color table with other than 8 bits for each primary color can be loaded.
CAPS_COLTABL_TRUE_MIX	1 if true mixing occurs when the logical color table has been realized, providing that the size of the logical color table is not greater than the number of distinct colors supported (see element CAPS_COLORS).
CAPS_COLTABL_REALIZE	1 if a loaded color table can be realized.

CAPS_MOUSE_BUTTONS

The number of pointing device buttons that are available. A returned value of 0 indicates that there are no pointing device buttons available.

CAPS_FOREGROUND_MIX_SUPPORT

Foreground mix support:

CAPS_FM_OR	Logical OR.
CAPS_FM_OVERPAINT	Overpaint.
CAPS_FM_XOR	Logical XOR.
CAPS_FM_LEAVEALONE	Leave alone.
CAPS_FM_AND	Logical AND.
CAPS_FM_GENERAL_BOOLEAN	All other mix modes; see "GpiSetMix" in <i>Graphics Programming Interface Programming Reference</i> .

The value returned is the sum of the values appropriate to the mixes supported. A device capable of supporting OR must, as a minimum, return CAPS_FM_OR + CAPS_FM_OVERPAINT + CAPS_FM_LEAVEALONE, signifying support for the mandatory mixes OR, overpaint, and leave-alone.

Note that these numbers correspond to the decimal representation of a bit string that is six bits long, with each bit set to 1 if the appropriate mode is supported.

Those mixes returned as supported are guaranteed for all primitive types. For more information, see "GpiSetMix" in *Graphics Programming Interface Programming Reference*.

CAPS_BACKGROUND_MIX_SUPPORT

Background mix support:

CAPS_BM_OR	Logical OR.
CAPS_BM_OVERPAINT	Overpaint.
CAPS_BM_XOR	Logical XOR.
CAPS_BM_LEAVEALONE	Leave alone.
CAPS_BM_AND	Logical AND.
CAPS_BM_GENERAL_BOOLEAN	All other mix modes; see "GpiSetMix" in <i>Graphics Programming Interface Programming Reference</i> .

CAPS_BM_SRCTRSPARENT

Provides a transparent overlay function by not copying pels from the source bit map to the output bit map if they match the presentation space background color.

CAPS_BM_DESTTRANSPARENT

Provides a transparent underlay function by copying only the pels that match the presentation space background color from the source bit map to the output bit map.

The value returned is the sum of the values appropriate to the mixes supported. A device must, as a minimum, return CAPS_BM_OVERPAINT + CAPS_BM_LEAVEALONE, signifying support for the mandatory background mixes overpaint, and leave-alone.

Note that these numbers correspond to the decimal representation of a bit string that is four bits long, with each bit set to 1 if the appropriate mode is supported.

Those mixes returned as supported are guaranteed for all primitive types. For more information, see "GpiSetMix" in *Graphics Programming Interface Programming Reference*.

CAPS_VIO_LOADABLE_FONTS

Number of fonts that can be loaded for VIO.

CAPS_WINDOW_BYTE_ALIGNMENT

Whether or not the client area of VIO windows should be byte-aligned:

CAPS_BYTE_ALIGN_REQUIRED

Must be byte-aligned.

CAPS_BYTE_ALIGN_RECOMMENDED

More efficient if byte-aligned, but not required.

CAPS_BYTE_ALIGN_NOT_REQUIRED

Does not matter whether byte-aligned.

CAPS_BITMAP_FORMATS

Number of bit-map formats supported by device.

CAPS_RASTER_CAPS

Capability for device raster operations:

CAPS_RASTER_BITBLT

1 if GpiBitBlt and GpiWCBitBlt is supported.

CAPS_RASTER_BANDING

1 if banding is supported

CAPS_RASTER_BITBLT_SCALING

1 if GpiBitBlt and GpiWCBitBlt with scaling is supported.

CAPS_RASTER_SET_PEL

1 if GpiSetPel is supported.

CAPS_RASTER_FONTS

1 if this device can draw raster fonts.

CAPS_RASTER_FLOOD_FILL

1 if GpiFloodFill

is supported.

CAPS_MARKER_HEIGHT
Default marker-box height in pels.

CAPS_MARKER_WIDTH
Default marker-box width in pels.

CAPS_DEVICE_FONTS
Number of device-specific fonts.

CAPS_GRAPHICS_SUBSET
Graphics drawing subset supported. (3 indicates GOCA DR/3)

CAPS_GRAPHICS_VERSION
Graphics architecture version number supported. (1 indicates Version 1)

CAPS_GRAPHICS_VECTOR_SUBSET
Graphics vector drawing subset supported. (2 indicates GOCA VS/2)

CAPS_DEVICE_WINDOWING
Device windowing support:

CAPS_DEV_WINDOWING_SUPPORT

1 if device
supports
windowing.

Other bits are
reserved 0.

CAPS_ADDITIONAL_GRAPHICS
Additional graphics support:

CAPS_GRAPHICS_KERNING_SUPPORT

1 if
device
supports
kerning.

CAPS_FONT_OUTLINE_DEFAULT

1 if
device
has a
default
outline
font.

CAPS_FONT_IMAGE_DEFAULT

1 if
device
has a
default
image
font.

CAPS_SCALED_DEFAULT_MARKERS

1 if
default
markers
are to be
scaled
by the
marker-
box
attribute.

CAPS_COLOR_CURSOR_SUPPORT

1 if
device
supports
colored
cursors.

CAPS_PALETTE_MANAGER

1 if
device
supports
palette
function

s (see "GpiCreatePalette" in the *Graphics Programming Interface Programming Reference*).

CAPS_COSMETIC_WIDELINE_SUPPORT

1 if device supports cosmetic thick lines (see "GpiSetLineWidth" in the *Graphics Programming Interface Programming Reference*).

Other bits are reserved 0.

CAPS_PHYS_COLORS

Maximum number of distinct colors available on the device.

CAPS_COLOR_INDEX

Maximum logical color-table index supported for this device. For the EGA and VGA drivers, the value is 63.

CAPS_GRAPHICS_CHAR_WIDTH

Default graphics character-box width, in pels.

CAPS_GRAPHICS_CHAR_HEIGHT

Default graphics character-box height, in pels.

CAPS_HORIZONTAL_FONT_RES

Effective horizontal device resolution in pels per inch, for the purpose of selecting fonts.

For printers, this is the actual device resolution, but for displays it may differ from the actual resolution for reasons of legibility.

CAPS_VERTICAL_FONT_RES

Effective vertical device resolution in pels per inch, for the purpose of selecting fonts.

CAPS_DEVICE_FONT_SIM

Identifies which simulations are valid on device fonts.

Valid flags are:

CAPS_DEV_FONT_SIM_BOLD
CAPS_DEV_FONT_SIM_ITALIC
CAPS_DEV_FONT_SIM_UNDERSCORE
CAPS_DEV_FONT_SIM_STRIKEOUT

CAPS_LINEWIDTH_THICK

Cosmetic thickness of lines and arcs on this device, when *fxLineWidth* is LINEWIDTH_THICK (see "GpiSetLineWidth" in the *Graphics Programming Interface Programming Reference*). The units are pels. A value

of 0 is interpreted as 2 pels.

CAPS_DEVICE_POLYSET_POINTS

Number of points in a polyset that a device can handle.

rc ([BOOL](#)) - returns
Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

DevQueryCaps - Remarks

GpiQueryDevice can be used to find the handle of the currently associated device context.

DevQueryCaps - Errors

Possible returns from [WinGetLastError](#)

PMERR_INV_HDC (0x207C)

An invalid device-context handle or (micro presentation space) presentation-space handle was specified.

PMERR_INV_QUERY_ELEMENT_NO (0x20BC)

An invalid start parameter was specified with DevQueryCaps.

PMERR_INV_LENGTH_OR_COUNT (0x2092)

An invalid length or count parameter was specified.

DevQueryCaps - Related Functions

Prerequisite Functions

- [DevOpenDC](#) (for CAPS_FAMILY)

Related Functions

- [DevQueryDeviceNames](#)
 - [DevQueryHardcopyCaps](#)
-

DevQueryCaps - Example Code

In this example the driver is queried to see if it supports input, output, or both. Note that a valid device context handle must be passed. This example assumes a DevOpenDC call has been made to obtain the device context handle.

```
#define INCL_DEV
#include <OS2.H>
```

```

HDC hdc;
LONG lStart;
LONG lCount;
BOOL flreturn;
LONG alArray[CAPS_TECHNOLOGY];
lCount = CAPS_TECHNOLOGY;
lStart = CAPS_FAMILY;

flreturn = DevQueryCaps(hdc,          /* device context handle */
                        lStart,        /* number of first item */
                        lCount,        /* count of items */
                        alArray);      /* array of longs which */
                                      /* will contain the return */
                                      /* information. */

switch(alArray[CAPS_IO_CAPS])        /* we test the CAPS_IO_CAPS */
{                                     /* element of the array to */
    case CAPS_IO_SUPPORTS_OP:        /* find out which options */
                                      /* are supported. */
                                      /* device supports output.*/

    break;

    case CAPS_IO_SUPPORTS_IP:        /* device supports input. */

    break;

    case CAPS_IO_SUPPORTS_IO:        /* device supports both */
                                      /* input and output. */

    break;
    default:
    break;
}

```

DevQueryCaps - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DevQueryDeviceNames

DevQueryDeviceNames - Syntax

This function causes a presentation driver to return the names, descriptions, and data types of the devices it supports.

```

#define INCL_DEV /* Or use INCL_PM, */
#include <os2.h>

```

```

HAB      hab;           /* Anchor-block handle. */
PSZ      pszDriverName; /* Fully-qualified name of the file containing the presentation driver. */
PLONG    pldn;          /* Maximum number of device names and descriptions that can be returned. */
PSTR32    aDeviceName;  /* Device-name array. */
PSTR64    aDeviceDesc;  /* Device-description array. */
PLONG     pldt;          /* Maximum number of data types that can be returned. */
PSTR16     aDataType;    /* Data type array. */
BOOL      rc;           /* Success indicator. */

rc = DevQueryDeviceNames(hab, pszDriverName,
                        pldn, aDeviceName, aDeviceDesc, pldt,
                        aDataType);

```

DevQueryDeviceNames Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

DevQueryDeviceNames Parameter - pszDriverName

pszDriverName (**PSZ**) - input
Fully-qualified name of the file containing the presentation driver.

The file-name extension is DRV.

DevQueryDeviceNames Parameter - pldn

pldn (**PLONG**) - in/out
Maximum number of device names and descriptions that can be returned.

On input, it must be greater or equal to 0. *pldn* can have the following values:

Zero	The number of device names and descriptions supported is returned; <i>aDeviceName</i> and <i>aDeviceDesc</i> are not updated.
Nonzero	<i>pldn</i> is updated to the number returned in <i>aDeviceName</i> and <i>aDeviceDesc</i> ; <i>aDeviceName</i> and <i>aDeviceDesc</i> are updated.

DevQueryDeviceNames Parameter - aDeviceName

aDeviceName (**PSTR32**) - output
Device-name array.

An array of null-terminated strings, each element of which identifies a particular device. Valid names are defined by presentation drivers.

DevQueryDeviceNames Parameter - aDeviceDesc

aDeviceDesc (**PSTR64**) - output
Device-description array.

An array of null-terminated strings, each element of which is a description of a particular device. Valid descriptions are defined by presentation drivers.

DevQueryDeviceNames Parameter - pldt

pldt (**PLONG**) - in/out
Maximum number of data types that can be returned.

On input, it must be greater or equal to 0. *pldt* can have the following values:

- | | |
|---------|--|
| Zero | The number of data types supported is returned, and <i>aDataType</i> is not updated. |
| Nonzero | <i>pldt</i> is updated to the number returned, and <i>aDataType</i> is updated. |

DevQueryDeviceNames Parameter - aDataType

aDataType (**PSTR16**) - output
Data type array.

An array of null-terminated strings, each element of which identifies a data type. Valid data types are defined by presentation drivers.

DevQueryDeviceNames Return Value - rc

rc (**BOOL**) - returns
Success indicator.

- | | |
|-------|-----------------------|
| TRUE | Successful completion |
| FALSE | Error occurred. |

DevQueryDeviceNames - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

pszDriverName ([PSZ](#)) - input
Fully-qualified name of the file containing the presentation driver.

The file-name extension is DRV.

pIdn ([PLONG](#)) - in/out
Maximum number of device names and descriptions that can be returned.

On input, it must be greater or equal to 0. *pIdn* can have the following values:

Zero	The number of device names and descriptions supported is returned; <i>aDeviceName</i> and <i>aDeviceDesc</i> are not updated.
Nonzero	<i>pIdn</i> is updated to the number returned in <i>aDeviceName</i> and <i>aDeviceDesc</i> ; <i>aDeviceName</i> and <i>aDeviceDesc</i> are updated.

aDeviceName ([PSTR32](#)) - output
Device-name array.

An array of null-terminated strings, each element of which identifies a particular device. Valid names are defined by presentation drivers.

aDeviceDesc ([PSTR64](#)) - output
Device-description array.

An array of null-terminated strings, each element of which is a description of a particular device. Valid descriptions are defined by presentation drivers.

pIdt ([PLONG](#)) - in/out
Maximum number of data types that can be returned.

On input, it must be greater or equal to 0. *pIdt* can have the following values:

Zero	The number of data types supported is returned, and <i>aDataType</i> is not updated.
Nonzero	<i>pIdt</i> is updated to the number returned, and <i>aDataType</i> is updated.

aDataType ([PSTR16](#)) - output
Data type array.

An array of null-terminated strings, each element of which identifies a data type. Valid data types are defined by presentation drivers.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DevQueryDeviceNames - Remarks

An application can first call this function with *pIdn* and *pIdt* set to 0 to find how much storage is needed for the data areas. Having allocated the storage, the application calls the function a second time for the data to be entered.

"HP Laserjet IID" is an example of a device name, "Hewlett-Packard Laserjet IID" is an example of a device description, and "PM_Q_STD" is an example of a data type.

DevQueryDeviceNames - Errors

Possible returns from [WinGetLastError](#)

PMERR_INV_LENGTH_OR_COUNT (0x2092)
An invalid length or count parameter was specified.

DevQueryDeviceNames - Related Functions

Related Functions

- [DevQueryCaps](#)
- [DevQueryHardcopyCaps](#)

DevQueryDeviceNames - Example Code

This example uses DevQueryDeviceNames to return the names, descriptions, and data types of supported devices for a presentation driver. The first call to DevQueryDeviceNames determines the number of names, description, and data types available; after allocating the arrays, the second call actually returns the information in the arrays.

```
#define INCL_DEV                /* Device Function definitions */
#define INCL_DOSMEMMGR         /* DOS Memory Manager Functions */
#include <os2.h>

BOOL fSuccess;                /* success indicator */
HAB hab;                      /* Anchor-block handle */
LONG pldn = 0L;               /* number of device names/descriptions */
LONG pldt = 0L;               /* number of data types */
PSTR32 aDeviceName;           /* array of device names */
PSTR64 aDeviceDesc;           /* array of device descriptions */
PSTR16 aDataType;             /* array of data types */

/* query number of supported names/descriptions/data types
(pldn & pldt both 0) */
fSuccess = DevQueryDeviceNames(hab, "IBM4201.DRV", &pldn,
                              aDeviceName, aDeviceDesc, &pldt,
                              aDataType);

if (fSuccess)
{
    /* allocate arrays */
    DosAllocMem((VOID *)aDeviceName, (ULONG)pldn*sizeof(STR32),
                PAG_COMMIT | PAG_WRITE);
    DosAllocMem((VOID *)aDeviceDesc, (ULONG)pldn*sizeof(STR64),
                PAG_COMMIT | PAG_WRITE);
    DosAllocMem((VOID *)aDataType, (ULONG)pldt*sizeof(STR16),
                PAG_COMMIT | PAG_WRITE);

    /* query supported device information */
    fSuccess = DevQueryDeviceNames(hab, "IBM4201.DRV", &pldn,
                                   aDeviceName, aDeviceDesc, &pldt,
                                   aDataType);
}
```

DevQueryDeviceNames - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

DevQueryHardcopyCaps

DevQueryHardcopyCaps - Syntax

This function queries the hard-copy capabilities of a device.

```
#define INCL_DEV /* Or use INCL_PM, */
#include <os2.h>

HDC      hdc;          /* Device-context handle. */
LONG     lStartForm;    /* Start-forms code. */
LONG     lForms;        /* Number of forms to query. */
PHCIINFO phciHcInfo;    /* Hard-copy capabilities information. */
LONG     lFormsReturned; /* Details of forms: */

lFormsReturned = DevQueryHardcopyCaps(hdc,
                                       lStartForm, lForms, phciHcInfo);
```

DevQueryHardcopyCaps Parameter - hdc

hdc ([HDC](#)) - input
Device-context handle.

DevQueryHardcopyCaps Parameter - lStartForm

IStartForm ([LONG](#)) - input
Start-forms code.

Forms-code number from which the query is to start. The first forms code has the value 0. *IStartForm* is used with *IForms*.

DevQueryHardcopyCaps Parameter - IForms

IForms ([LONG](#)) - input
Number of forms to query.

If 0, the number of forms codes defined is returned. If greater than zero, this function returns the number of forms codes for which information is returned.

For example, if there are five forms codes defined, and *IStartForm* = 2 and *IForms* = 3, a query is performed for forms codes 2, 3, and 4. The result is returned in the buffer pointed to by *phciHcInfo*.

DevQueryHardcopyCaps Parameter - phciHcInfo

phciHcInfo ([PHCINFO](#)) - output
Hard-copy capabilities information.

A buffer containing the results of the query. The result consists of *IForms* copies of the [HCINFO](#) structure.

At least one of the defined forms codes must have the HCAPS_CURRENT bit set. There might be more than one with either the HCAPS_CURRENT or the HCAPS_SELECTABLE bits set.

For a job to be selected by the spooler for printing, each one of the forms specified in the FORM spooler parameter (see *pszSpoolerParams* in [DEVOPENSTRUC](#)) must be either HCAPS_CURRENT or HCAPS_SELECTABLE. In other cases, the spooler holds the job with a "forms mismatch" error.

DevQueryHardcopyCaps Return Value - IFormsReturned

IFormsReturned ([LONG](#)) - returns
Details of forms:

DQHC_ERROR
Error.

>=0
If *IForms* equals 0, number of forms available.
If *IForms* does not equal 0, number of forms returned.

DevQueryHardcopyCaps - Parameters

hdc ([HDC](#)) - input
Device-context handle.

IStartForm ([LONG](#)) - input
Start-forms code.

Forms-code number from which the query is to start. The first forms code has the value 0. *IStartForm* is used with *IForms*.

IForms ([LONG](#)) - input
Number of forms to query.

If 0, the number of forms codes defined is returned. If greater than zero, this function returns the number of forms codes for which information is returned.

For example, if there are five forms codes defined, and *IStartForm* = 2 and *IForms* = 3, a query is performed for forms codes 2, 3, and 4. The result is returned in the buffer pointed to by *phciHcInfo*.

phciHcInfo ([PHCINFO](#)) - output
Hard-copy capabilities information.

A buffer containing the results of the query. The result consists of *IForms* copies of the [HCINFO](#) structure.

At least one of the defined forms codes must have the HCAPS_CURRENT bit set. There might be more than one with either the HCAPS_CURRENT or the HCAPS_SELECTABLE bits set.

For a job to be selected by the spooler for printing, each one of the forms specified in the FORM spooler parameter (see *pszSpoolerParams* in [DEVOPENSTRUC](#)) must be either HCAPS_CURRENT or HCAPS_SELECTABLE. In other cases, the spooler holds the job with a "forms mismatch" error.

IFormsReturned ([LONG](#)) - returns
Details of forms:

DQHC_ERROR
Error.

>=0
If *IForms* equals 0, number of forms available.
If *IForms* does not equal 0, number of forms returned.

DevQueryHardcopyCaps - Errors

Possible returns from [WinGetLastError](#)

PMERR_INV_HDC (0x207C)
An invalid device-context handle or (micro presentation space) presentation-space handle was specified.

PMERR_INV_FORMS_CODE (0x2076)
An invalid forms code parameter was specified with DevQueryHardcopyCaps.

PMERR_INV_LENGTH_OR_COUNT (0x2092)
An invalid length or count parameter was specified.

DevQueryHardcopyCaps - Related Functions

Prerequisite Functions

- [DevOpenDC](#)

Related Functions

- [DevQueryCaps](#)
- [DevQueryDeviceNames](#)

DevQueryHardcopyCaps - Example Code

The height and width of the capability of the output device is queried for each form code available. Note that a valid device context handle must be passed. This example assumes a DevOpenDC call has been made to obtain the device context handle of a say a printer.

```
#define INCL_DEV
#include <OS2.H>

HDC hdc;
LONG lStartForm;      /* Form code number from which the query */
                     /* is to start */
LONG lForms;          /* number of forms to query */
/* array of structures containing return information. */
HCINFO ahciHcInfo[5];
LONG lreturn;
int i;
HCINFO height[5];
HCINFO width[5];

lStartForm = 0L;
lForms = 0L;          /* the actual number of forms codes is */
                     /* returned. There will be lreturn */
                     /* copies of the HINFO structure. */

lreturn = DevQueryHardcopyCaps(hdc,
                               lStartForm,
                               lForms,
                               ahciHcInfo);

if (lreturn > 5)
{
    lreturn = 5L;      /* we only want the first five form codes */
}                    /* if there are more than five */

for(i = 0; i < lreturn; i++)
{
    width[lreturn].cx = ahciHcInfo[lreturn].cx;
    height[lreturn].cy = ahciHcInfo[lreturn].cy;
}
```

DevQueryHardcopyCaps - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

Direct Manipulation Functions

This section describes functions that an application would use to initiate or participate in a direct manipulation operation.

DrgAcceptDroppedFiles

DrgAcceptDroppedFiles - Syntax

This function handles the file direct manipulation protocol for a given window.

```
#define INCL_WINSTDDRAG
#include <os2.h>

HWND    hwnd;          /* Handle of calling window. */
PSZ     pszPath;       /* Directory in which to place the dropped files. */
PSZ     pszTypes;      /* List of types that are acceptable to the drop. */
ULONG   ulDefaultOp;   /* Default drag operation for this window. */
ULONG   ulRsvd;        /* Reserved. */
BOOL    rc;            /* Success indicator. */

rc = DrgAcceptDroppedFiles(hwnd, pszPath,
                           pszTypes, ulDefaultOp, ulRsvd);
```

DrgAcceptDroppedFiles Parameter - hwnd

hwnd (**HWND**) - input
Handle of calling window.

DrgAcceptDroppedFiles Parameter - pszPath

pszPath (**PSZ**) - input
Directory in which to place the dropped files.

If NULL, the files are placed in the current directory.

DrgAcceptDroppedFiles Parameter - pszTypes

pszTypes (**PSZ**) - input
List of types that are acceptable to the drop.

This string is of the form:

type[, type . . .]

When this pointer is NULL, any type of file will be accepted.

DrgAcceptDroppedFiles Parameter - ulDefaultOp

ulDefaultOp (**ULONG**) - input
Default drag operation for this window.

The operation is either DO_MOVE or DO_COPY.

DrgAcceptDroppedFiles Parameter - ulRsvd

ulRsvd (**ULONG**) - input
Reserved.

DrgAcceptDroppedFiles Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgAcceptDroppedFiles - Parameters

hwnd (**HWND**) - input
Handle of calling window.

pszPath (**PSZ**) - input
Directory in which to place the dropped files.

If NULL, the files are placed in the current directory.

pszTypes ([PSZ](#)) - input

List of types that are acceptable to the drop.

This string is of the form:

```
type[ , type. . . ]
```

When this pointer is NULL, any type of file will be accepted.

uiDefaultOp ([ULONG](#)) - input

Default drag operation for this window.

The operation is either DO_MOVE or DO_COPY.

uiRsvd ([ULONG](#)) - input

Reserved.

rc ([BOOL](#)) - returns

Success indicator.

TRUE

Successful completion.

FALSE

Error occurred.

DrgAcceptDroppedFiles - Remarks

This function handles the file direct manipulation protocol for a given window. The window responds ([DOR_DROP](#), *usDefaultOp*) to [DM_DRAGOVER](#) messages for items with a type matching the acceptable type string and with a rendering mechanism and format of <DRM_OS2FILE,DRF_UNKNOWN>. Not all dragged objects must match this criteria for the drop to be acceptable.

After the drop occurs, this function handles the conversation required to complete the direct manipulation operation for all acceptable objects. A [DM_ENDCONVERSATION](#) ([DMFL_TARGETFAIL](#)) message is sent to the source when an object is unacceptable.

When an error occurs during a move or copy, the caller is sent a [DM_DRAGERROR](#) message. The caller can take corrective action.

As the move or copy operation is successfully completed for each file, a [DM_DRAGFILECOMPLETE](#) message is sent to the caller. No message is sent when the operation fails.

The function returns TRUE if the operation is successful and FALSE if an error occurs.

DrgAcceptDroppedFiles - Related Functions

Related Functions

- [DrgDragFiles](#)

DrgAcceptDroppedFiles - Example Code

This example uses the [DrgAcceptDroppedFiles](#) function to define the direct manipulation protocol of the given window, accept all file types, and use the current directory as the drop directory.


```

#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

BOOL    fSuccess;        /* Indicate success or failure */
HWND    Hwnd;            /* Handle of calling window */
PSZ     pszPath;         /* Directory in which to place the
                        /* dropped files
PSZ     pszTypes;        /* A list of types that are acceptable
ULONG   ulDefaultOp;     /* Default drag operation

pszPath = NULL;          /* Drop file in current directory
pszTypes = NULL;         /* Accept any file type
ulDefaultOp = DO_MOVE;   /* Default drag operation is move

fSuccess = DrgAcceptDroppedFiles(Hwnd, pszPath, pszTypes,
                                ulDefaultOp, 0);

```

DrgAcceptDroppedFiles - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgAccessDraginfo

DrgAccessDraginfo - Syntax

This function accesses a [DRAGINFO](#) structure.

```

#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGINFO    pdinfo; /* Pointer to the DRAGINFO structure. */
BOOL         rc;     /* Success indicator. */

rc = DrgAccessDraginfo(pdinfo);

```

DrgAccessDraginfo Parameter - pdinfo

pdinfo ([PDRAGINFO](#)) - input
Pointer to the [DRAGINFO](#) structure.

DrgAccessDraginfo Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgAccessDraginfo - Parameters

pdinfo ([PDRAGINFO](#)) - input
Pointer to the [DRAGINFO](#) structure.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgAccessDraginfo - Remarks

This function is used by the target of a drag operation to access a [DRAGINFO](#) structure. The address of the structure is passed in a drag message ([DM_DRAGOVER](#), [DM_DROP](#), or [DM_DROPHELP](#)).

To release the structure, use the [DrgFreeDraginfo](#) function.

DrgAccessDraginfo - Errors

Possible returns from [WinGetLastError](#)

PMERR_ACCESS_DENIED (0x150D)	The memory block was not allocated properly.
------------------------------	--

DrgAccessDraginfo - Related Functions

Related Functions

- [DrgAllocDraginfo](#)
- [DrgDrag](#)
- [DrgFreeDraginfo](#)
- [DrgPushDraginfo](#)

DrgAccessDraginfo - Example Code

This example uses the DrgAccessDraginfo function to make an existing drag information structure (created by the [DrgAllocDraginfo](#) function) available.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

BOOL      fSuccess;      /* Indicate success or failure      */
DRAGINFO  Draginfo;      /* Drag-information structure      */

fSuccess = DrgAccessDraginfo(&Draginfo);
```

DrgAccessDraginfo - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgAddStrHandle

DrgAddStrHandle - Syntax

This function creates a handle to a string.

```
#define INCL_WINSTDDRAG
#include <os2.h>

PSZ      pString; /* String for which a handle is to be created. */
HSTR     hstr;    /* String handle. */
```

```
hstr = DrgAddStrHandle(pString);
```

DrgAddStrHandle Parameter - pString

pString ([PSZ](#)) - input
String for which a handle is to be created.

DrgAddStrHandle Return Value - hstr

hstr ([HSTR](#)) - returns
String handle.

NULLHANDLE	Error occurred.
Other	String handle created.

DrgAddStrHandle - Parameters

pString ([PSZ](#)) - input
String for which a handle is to be created.

hstr ([HSTR](#)) - returns
String handle.

NULLHANDLE	Error occurred.
Other	String handle created.

DrgAddStrHandle - Remarks

The handle can be used by any application to reference the input string.

This function must be called by the source of a drag whenever a string is to be passed in a [DRAGINFO](#) structure.

DrgAddStrHandle - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_PARAMETERS (0x1208)

An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32 768 to +32 767 cannot be converted to a [SHORT](#), and a negative number cannot be converted to a [ULONG](#) or [USHORT](#).

PMERR_RESOURCE_DEPLETION (0x20F9)

An internal resource depletion error has occurred.

DrgAddStrHandle - Related Functions

Related Functions

- [DrgDeleteStrHandle](#)
 - [DrgQueryStrName](#)
-

DrgAddStrHandle - Example Code

This example calls the DrgAddStrHandle function to create handles for strings that are used in a [DRAGITEM](#) structure.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

USHORT ID_ITEM = 1; /* Drag item identifier */
HWND hwnd; /* Window handle */
DRAGITEM ditem; /* DRAGITEM structure */

/* Initialize the DRAGITEM structure */
ditem.hwndItem = hwnd; /* Conversation partner */
ditem.ulItemID = ID_ITEM; /* Identifies item being dragged */
ditem.hstrType = DrgAddStrHandle(DRT_TEXT); /* Item is text */
ditem.hstrRMF = DrgAddStrHandle("<DRM_OS2FILE,DRF_TEXT>");
ditem.hstrContainerName = DrgAddStrHandle("C:\\");
ditem.hstrSourceName = DrgAddStrHandle("C:\\CONFIG.SYS");
ditem.hstrTargetName = DrgAddStrHandle("C:\\OS2\\CONFIG.SYS");
ditem.cxOffset = 0; /* X-offset of the origin of the */
/* image from the pointer hotspot*/
ditem.cyOffset = 0; /* Y-offset of the origin of the */
/* image from the pointer hotspot*/
ditem.fsControl = 0; /* Source item control flags */
/* object is open */
ditem.fsSupportedOps = 0;
```

DrgAddStrHandle - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgAllocDraginfo

DrgAllocDraginfo - Syntax

This function allocates a [DRAGINFO](#) structure.

```
#define INCL_WINSTDDRAG
#include <os2.h>

ULONG      cDitem;    /* Number of objects being dragged. */
PDRAGINFO  Draginfo;  /* Pointer to the DRAGINFO structure. */

Draginfo = DrgAllocDraginfo(cDitem);
```

DrgAllocDraginfo Parameter - cDitem

cDitem ([ULONG](#)) - input
Number of objects being dragged.

This number must be greater than 0.

DrgAllocDraginfo Return Value - Draginfo

Draginfo ([PDRAGINFO](#)) - returns
Pointer to the [DRAGINFO](#) structure.

NULL	Error occurred.
Other	The DRAGINFO structure.

DrgAllocDraginfo - Parameters

cDitem ([ULONG](#)) - input
Number of objects being dragged.

This number must be greater than 0.

Draginfo ([PDRAGINFO](#)) - returns
Pointer to the [DRAGINFO](#) structure.

NULL	Error occurred.
Other	The DRAGINFO structure.

DrgAllocDraginfo - Remarks

This function must be called before the [DrgDrag](#) function is called.

The caller can define a default operation for the objects represented by the [DRAGINFO](#) structure by modifying the *usOperation* field. If the *usOperation* field is modified, the new value will be sent to the target as the operation whenever a DO_DEFAULT operation would normally be sent. The caller should not modify any other part of the [DRAGINFO](#) structure. The [DRAGITEM](#) structures associated with the [DRAGINFO](#) structure should only be altered with [DrgSetDragitem](#) or by using a pointer obtained with [DrgQueryDragitemPtr](#).

DrgAllocDraginfo - Errors

Possible returns from [WinGetLastError](#)

PMERR_INSUFFICIENT_MEMORY (0x203E)
The operation terminated through insufficient memory.

PMERR_INVALID_PARAMETERS (0x1208)
An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32 768 to +32 767 cannot be converted to a [SHORT](#), and a negative number cannot be converted to a [ULONG](#) or [USHORT](#).

DrgAllocDraginfo - Related Functions

Related Functions

- [DrgAccessDraginfo](#)
- [DrgDrag](#)
- [DrgFreeDraginfo](#)
- [DrgPushDraginfo](#)

DrgAllocDraginfo - Example Code

This example calls the [DrgAllocDraginfo](#) function to create a Drag structure for a single object and uses the new structure to set the [DRAGITEM](#) ([DrgSetDragitem](#)) structure.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

PDRAGINFO pdinfo; /* Pointer to DRAGINFO structure */
HWND hwnd; /* Handle of calling (source) window */
BOOL flResult; /* Result indicator */
```

```
DRAGITEM    ditem;          /* DRAGITEM structure          */

pdinfo = DrgAllocDraginfo(1); /* Create the DRAGINFO structure */
                        /* Set the drag item          */
flResult= DrgSetDragitem(pdinfo, &ditem, (ULONG)sizeof(ditem), 0);
```

DrgAllocDraginfo - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgAllocDragtransfer

DrgAllocDragtransfer - Syntax

This function allocates a specified number of [DRAGTRANSFER](#) structures from a single segment.

```
#define INCL_WINSTDDRAG
#include <os2.h>

ULONG          cdxfer;      /* Number of DRAGTRANSFER structures to be allocated. */
PDRAGTRANSFER  pDragtransfer; /* Pointer to an array of DRAGTRANSFER structures. */

pDragtransfer = DrgAllocDragtransfer(cdxfer);
```

DrgAllocDragtransfer Parameter - cdxfer

cdxfer ([ULONG](#)) - input

Number of [DRAGTRANSFER](#) structures to be allocated.

This number must be greater than 0.

DrgAllocDragtransfer Return Value - pDragtransfer

pDragtransfer ([PDRAGTRANSFER](#)) - returns
Pointer to an array of [DRAGTRANSFER](#) structures.

NULL	Error occurred.
Other	The array of DRAGTRANSFER structures.

DrgAllocDragtransfer - Parameters

cdxfer ([ULONG](#)) - input
Number of [DRAGTRANSFER](#) structures to be allocated.

This number must be greater than 0.

pDragtransfer ([PDRAGTRANSFER](#)) - returns
Pointer to an array of [DRAGTRANSFER](#) structures.

NULL	Error occurred.
Other	The array of DRAGTRANSFER structures.

DrgAllocDragtransfer - Remarks

This function must be called before sending a [DM_RENDER](#) message.

DrgAllocDragtransfer - Errors

Possible returns from [WinGetLastError](#)

[PMERR_MEMORY_ALLOCATION_ERR](#) (0x1112)
An error occurred during memory management.

[PMERR_INSUFFICIENT_MEMORY](#) (0x203E)
The operation terminated through insufficient memory.

[PMERR_PARAMETER_OUT_OF_RANGE](#) (0x1003)
The value of a parameter was not within the defined valid range for that parameter.

DrgAllocDragtransfer - Related Functions

- Related Functions**
- [DrgFreeDragtransfer](#)

- [DrgSendTransferMsg](#)

DrgAllocDragtransfer - Example Code

This example calls the DrgAllocDragtransfer function to allocate a single [DRAGTRANSFER](#) structure and adds a pointer to a [DRAGITEM](#) structure for an object that will be transferred.

```
#define INCL_WINSTDDRAG    /* Direct Manipulation (Drag) Functions*/
#include <os2.h>

PDRAGTRANSFER  pResult;    /* Pointer to DRAGTRANSFER structure    */
PDRAGITEM       pDragitem; /* Pointer to DRAGITEM structure    */

pResult = DrgAllocDragtransfer(1);

if (pResult != NULL)        /* Indicate DRAGITEM to be transferred */
    pResult->pditem = pDragitem;
```

DrgAllocDragtransfer - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgCancelLazyDrag

DrgCancelLazyDrag - Syntax

This function is called to cancel the current drag operation.

```
#define INCL_WINSTDDRAG
#include <os2.h>

BOOL    rc; /* Success indicator. */

rc = DrgCancelLazyDrag();
```

DrgCancelLazyDrag Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

Possible values are described in the following list:

TRUE	Lazy drag is successfully canceled.
FALSE	An error occurred.

DrgCancelLazyDrag - Parameters

rc ([BOOL](#)) - returns
Success indicator.

Possible values are described in the following list:

TRUE	Lazy drag is successfully canceled.
FALSE	An error occurred.

DrgCancelLazyDrag - Remarks

This function posts the [DM_DROPNOTIFY](#) message to the source window, specifying a target window handle of zero in *hwndTarget*. The source window must then free [DRAGINFO](#) using [DrgFreeDraginfo](#).

DrgCancelLazyDrag - Example Code

This example shows the canceling of a lazy drag operation.

```
#define INCL_WINSTDDRAG
#include <os2.h>

BOOL      bResult;          /* Return code from API */

bResult=DrgCancelLazyDrag();
```

DrgCancelLazyDrag - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

DrgDeleteDraginfoStrHandles

DrgDeleteDraginfoStrHandles - Syntax

This function deletes each unique string handle in a [DRAGINFO](#) structure.

```
#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGINFO    pDraginfo; /* Pointer to the DRAGINFO structure that contains string handles to delete. */
BOOL         rc;        /* Success indicator. */

rc = DrgDeleteDraginfoStrHandles(pDraginfo);
```

DrgDeleteDraginfoStrHandles Parameter - pDraginfo

pDraginfo ([PDRAGINFO](#)) - input
Pointer to the [DRAGINFO](#) structure that contains string handles to delete.

DrgDeleteDraginfoStrHandles Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgDeleteDraginfoStrHandles - Parameters

pDraginfo ([PDRAGINFO](#)) - input
Pointer to the [DRAGINFO](#) structure that contains string handles to delete.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgDeleteDraginfoStrHandles - Remarks

Using this function is equivalent to calling the [DrgDeleteStrHandle](#) function for each unique string in a [DRAGINFO](#) structure.

This function must be called by the target of a direct manipulation operation either:

- After processing a [DM_DROPHELP](#) message
or
- After completing the direct manipulation operation begun as a result of a [DM_DROP](#) message.

DrgDeleteDraginfoStrHandles - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_PARAMETERS (0x1208)
An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32 768 to +32 767 cannot be converted to a [SHORT](#), and a negative number cannot be converted to a [ULONG](#) or [USHORT](#).

DrgDeleteDraginfoStrHandles - Related Functions

Related Functions

- [DrgDeleteStrHandle](#)

DrgDeleteDraginfoStrHandles - Example Code

This example calls the [DrgDeleteDraginfoStrHandles](#) function to delete all unique string handles associated with the specified [DRAGINFO](#) structure (previously allocated by the [DrgAllocDraginfo](#) function).

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

BOOL      fSuccess;      /* Indicate success or failure */
DRAGINFO  Draginfo;      /* DRAGINFO structure containing string */
                        /* handles to delete */
```

```
fSuccess = DrgDeleteDraginfoStrHandles (&Draginfo);
```

DrgDeleteDraginfoStrHandles - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

DrgDeleteStrHandle

DrgDeleteStrHandle - Syntax

This function deletes a string handle.

```
#define INCL_WINSTDDRAG
#include <os2.h>

HSTR    hstr; /* The string handle to delete. */
BOOL    rc;   /* Success indicator. */

rc = DrgDeleteStrHandle(hstr);
```

DrgDeleteStrHandle Parameter - hstr

hstr ([HSTR](#)) - input
The string handle to delete.

DrgDeleteStrHandle Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgDeleteStrHandle - Parameters

hstr ([HSTR](#)) - input
The string handle to delete.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgDeleteStrHandle - Remarks

This function must be used to delete a string handle created by the [DrgAddStrHandle](#) function.

DrgDeleteStrHandle - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_PARAMETERS (0x1208)

DrgDeleteStrHandle - Related Functions

Related Functions

- [DrgAddStrHandle](#)
- [DrgDeleteDraginfoStrHandles](#)

DrgDeleteStrHandle - Example Code

This example calls the DrgDeleteStrHandle function to delete an existing string handle (returned by a previous call to the [DrgAddStrHandle](#)

function).

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

BOOL fSuccess;          /* Indicate success or failure */
HSTR Hstr;              /* String handle */

fSuccess = DrgDeleteStrHandle (Hstr);
```

DrgDeleteStrHandle - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgDrag

DrgDrag - Syntax

This function performs a drag operation.

```
#define INCL_WINSTDDRAG
#include <os2.h>

HWND      hwndSource; /* Handle of the source window calling this function. */
PDRAGINFO pdinfo;    /* Pointer to the DRAGINFO structure. */
PDRAGIMAGE pdimg;     /* Pointer to an array of DRAGIMAGE structures. */
ULONG      cdimg;     /* Number of DRAGIMAGE structures in the pdimg array. Must be > 0. */
LONG       vkTerminate; /* Pointing device button that ends the drag operation. */
PVOID      pRsvd;     /* Reserved value, must be NULL. */
HWND       hwndDest;  /* Handle of window on which the dragged objects were dropped. */

hwndDest = DrgDrag(hwndSource, pdinfo, pdimg,
                   cdimg, vkTerminate, pRsvd);
```

DrgDrag Parameter - hwndSource

hwndSource ([HWND](#)) - input
Handle of the source window calling this function.

DrgDrag Parameter - pdinfo

pdinfo ([PDRAGINFO](#)) - in/out
Pointer to the [DRAGINFO](#) structure.

DrgDrag Parameter - pdimg

pdimg ([PDRAGIMAGE](#)) - input
Pointer to an array of [DRAGIMAGE](#) structures.

These structures describe the images that are to be drawn under the direct manipulation pointer during the drag.

DrgDrag Parameter - cdimg

cdimg ([ULONG](#)) - input
Number of [DRAGIMAGE](#) structures in the *pdimg* array. Must be > 0.

DrgDrag Parameter - vkTerminate

vkTerminate ([LONG](#)) - input
Pointing device button that ends the drag operation.

Possible values are described in the following list:

VK_BUTTON1	Release of button 1 ends the drag.
VK_BUTTON2	Release of button 2 ends the drag.
VK_BUTTON3	Release of button 3 ends the drag.
VK_ENDDRAG	Release of the system-defined direct manipulation button ends the drag. This is the recommended value if the DrgDrag function call is invoked in response to a WM_BEGINDRAG message.

DrgDrag Parameter - pRsvd

pRsvd ([PVOID](#)) - input
Reserved value, must be NULL.

DrgDrag Return Value - hwndDest

hwndDest ([HWND](#)) - returns
Handle of window on which the dragged objects were dropped.

A return value of NULL indicates that an error occurred.

DrgDrag - Parameters

hwndSource ([HWND](#)) - input
Handle of the source window calling this function.

pdinfo ([PDRAGINFO](#)) - in/out
Pointer to the [DRAGINFO](#) structure.

pdimg ([PDRAGIMAGE](#)) - input
Pointer to an array of [DRAGIMAGE](#) structures.

These structures describe the images that are to be drawn under the direct manipulation pointer during the drag.

cdimg ([ULONG](#)) - input
Number of [DRAGIMAGE](#) structures in the *pdimg* array. Must be > 0.

vkTerminate ([LONG](#)) - input
Pointing device button that ends the drag operation.

Possible values are described in the following list:

VK_BUTTON1	Release of button 1 ends the drag.
VK_BUTTON2	Release of button 2 ends the drag.
VK_BUTTON3	Release of button 3 ends the drag.
VK_ENDDRAG	Release of the system-defined direct manipulation button ends the drag. This is the recommended value if the DrgDrag function call is invoked in response to a WM_BEGINDRAG message.

pRsvd ([PVOID](#)) - input
Reserved value, must be NULL.

hwndDest ([HWND](#)) - returns
Handle of window on which the dragged objects were dropped.

A return value of NULL indicates that an error occurred.

DrgDrag - Remarks

This function:

- Captures the mouse to the current thread
- Initiates a direct manipulation operation
- Uses the [DRAGIMAGE](#) structure to provide visual feedback to the user during the drag operation
- Notifies other windows as the drag object passes over
- Notifies the destination if the object is dropped
- Releases mouse capture.

Note: DrgDrag will fail if it is unable to capture the mouse. For example, if another window in the same thread has already set the capture.

Before invoking DrgDrag, the caller is responsible for:

- Obtaining a [DRAGINFO](#) structure using [DrgAllocDraginfo](#)
- Initializing the [DRAGITEM](#) structures using [DrgSetDragitem](#).

DrgDrag is called when the system-defined direct-manipulation button is pressed while the pointer is over a window and a pointing device movement follows. As the pointer moves over a potential target, a [DM_DRAGOVER](#) message is sent to the target. When the pointer moves from one target window to another, a [DM_DRAGLEAVE](#) message is sent to the former target.

If the pointer is over a valid target when the direct-manipulation button is released, a [DM_DROP](#) message is sent to the target.

Before the [DM_DROP](#) message is sent, the *cxOffset* and *cyOffset* fields are copied from the [DRAGIMAGE](#) structures to the corresponding fields in the [DRAGITEM](#) structures. The values from the first [DRAGIMAGE](#) are copied to the first [DRAGITEM](#), from the second [DRAGIMAGE](#) to the second [DRAGITEM](#), and so on. The target can use this information to place the images in the same spatial relationship after the drop. If there are more [DRAGITEM](#) structures than there are [DRAGIMAGE](#) structures, the *cxOffset* and *cyOffset* from the final [DRAGIMAGE](#) are placed in each of the remaining [DRAGITEM](#) structures.

The caller can define a default operation for the objects represented by the [DRAGINFO](#) structure by modifying the *usOperation* field. If the *usOperation* field is modified, the new value will be sent to the target as the operation whenever a DO_DEFAULT operation would normally be sent. The caller should not modify any other part of the [DRAGINFO](#) structure. The [DRAGITEM](#) structures associated with the [DRAGINFO](#) structure should only be altered with [DrgSetDragitem](#) or by using a pointer obtained with [DrgQueryDragitemPtr](#).

The following keys are active during the drag operation:

Esc	The drag operation is canceled.
F1	A DM_DROPHELP message is posted to the target so that it can provide context help for the drag operation. The drag operation is canceled.

Once the drag is commenced by calling DrgDrag, neither the image under the pointer nor the objects that comprise the drag set can be modified without canceling the drag and restarting.

On return from DrgDrag, the caller must free the [DRAGINFO](#) structure using [DrgFreeDraginfo](#).

If the dragged objects are not dropped, NULL is returned.

DrgDrag - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_INVALID_PARAMETERS (0x1208)
An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32 768 to +32 767 cannot be converted to a [SHORT](#), and a negative number cannot be converted to a [ULONG](#) or [USHORT](#).

PMERR_INSUFFICIENT_MEMORY (0x203E)
The operation terminated through insufficient memory.

DrgDrag - Related Functions

Prerequisite Functions

- [DrgAllocDraginfo](#)

Related Functions

- [DrgFreeDraginfo](#)
- [DrgLazyDrag](#)
- [DrgQueryDragitemPtr](#)
- [DrgSetDragitem](#)

DrgDrag - Example Code

This example uses the DrgDrag function to drag a single object in response to the direct-manipulation button being pressed while the pointer is over a drag object. The example shows the initialization of the [DRAGITEM](#), [DRAGINFO](#), and [DRAGIMAGE](#) structures used by the DrgDrag function.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#define INCL_WININPUT  /* Window Input Functions */
#include <os2.h>

PDRAGINFO pdinfo; /* Pointer to DRAGINFO structure */
HWND hwnd; /* Handle of calling (source) window */
BOOL flResult; /* Result indicator */
DRAGITEM ditem; /* DRAGITEM structure */
DRAGIMAGE dimg; /* DRAGIMAGE structure */
HBITMAP hbm; /* Bit-map handle */
HWND hwndDrop; /* Handle of drop (target) window */

case WM_BEGINDRAG:

    /******
    /* Initialize the DRAGITEM structure
    /******
    ditem.hwndItem = hwnd; /* Conversation partner */
    ditem.ulItemID = ID_ITEM; /* Identifies item being dragged */
    ditem.hstrType = DrgAddStrHandle(DRT_TEXT); /* Text item */
    ditem.hstrRMF = DrgAddStrHandle("<DRM_OS2FILE,DRF_TEXT>");
    ditem.hstrContainerName = DrgAddStrHandle("C:\\");
    ditem.hstrSourceName = DrgAddStrHandle("C:\\\\CONFIG.SYS");
    ditem.hstrTargetName = DrgAddStrHandle("C:\\OS2\\CONFIG.SYS");
    ditem.cxOffset = 0; /* X-offset of the origin of */
    /* the image from the pointer */
    /* hotspot */
    ditem.cyOffset = 0; /* Y-offset of the origin of */
    /* the image from the pointer */
    /* hotspot */
    ditem.fsControl = 0; /* Source item control flags */
    /* object is open */
    ditem.fsSupportedOps = 0;

    /******
    /* Create the DRAGINFO structure
    /******
    pdinfo = DrgAllocDraginfo(1);
    if (!pdinfo) return (FALSE); /* If allocation fails, */
    /* return FALSE */

    /******
    /* Initialize the DRAGIMAGE structure
    /******
    dimg.cb = sizeof(DRAGIMAGE); /* Size control block */
    dimg.cptl = 0;
    dimg.hImage = hbm; /* Image handle passed to */
    /* DrgDrag */
    dimg.sizlStretch.cx = 20L; /* Size to stretch ico or bmp to */
    dimg.sizlStretch.cy = 20L;
```

```

dimg.fl = DRG_BITMAP |          /* Flags passed to DrgDrag      */
          DRG_STRETCH;          /* Stretch to size specified  */
                                /* in sizlStretch             */
dimg.cxOffset = 0;              /* Offset of the origin of    */
dimg.cyOffset = 0;              /* the image from the pointer  */
                                /* hotspot                    */
/*****
/* Set the drag item
*****/
flResult= DrgSetDragitem(pdinfo, &ditem, (ULONG)sizeof(ditem),
                        0);

/*****
/* Perform the drag operation:
/* - Give the user a visual cue by changing the pointer to a
/*   bit map
/* - Send DM_DRAGOVER messages to the target window (in this
/*   case it is also the source)
/* NOTE: DrgDrag will fail if another window in the same
/*   thread already has the capture.
*****/
hwndDrop = DrgDrag(hwnd,        /* Source of the drag          */
                  pdinfo,       /* Pointer to DRAGINFO structure */
                  (PDRAGIMAGE)&dimg, /* Drag image                  */
                  1,             /* Size of the pdimg array     */
                  VK_ENDDRAG,    /* Release of direct-manipulation */
                                /* button ends the drag        */
                  NULL);         /* Reserved                    */

```

DrgDrag - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgDragFiles

DrgDragFiles - Syntax

This function begins a direct manipulation operation for one or more files.

```

#define INCL_WINSTDDRAG
#include <os2.h>

HWND      Hwnd;          /* Handle of calling window. */
PAPSZ     pFiles;        /* The names of the files to be dragged. */
PAPSZ     pTypes;        /* The file types of the files to be dragged. */
PAPSZ     pTargets;      /* Target file names. */

```

```
ULONG      cFiles;           /* Number of files to be dragged. */
HPOINTER   hptrDrag;        /* Icon to display during the drag. */
ULONG      vkTerm;          /* Button that ends the drag. */
BOOL       fSourceRender;    /* Flag indicating whether the source must perform the move or copy. */
ULONG      ulReserved;       /* Reserved. */
BOOL       rc;               /* Success indicator. */

rc = DrgDragFiles(Hwnd, pFiles, pTypes, pTargets,
                  cFiles, hptrDrag, vkTerm, fSourceRender,
                  ulReserved);
```

DrgDragFiles Parameter - Hwnd

Hwnd ([HWND](#)) - input
Handle of calling window.

DrgDragFiles Parameter - pFiles

pFiles ([PAPSZ](#)) - input
The names of the files to be dragged.

DrgDragFiles Parameter - pTypes

pTypes ([PAPSZ](#)) - input
The file types of the files to be dragged.

DrgDragFiles Parameter - pTargets

pTargets ([PAPSZ](#)) - input
Target file names.

DrgDragFiles Parameter - cFiles

cFiles ([ULONG](#)) - input
Number of files to be dragged.

DrgDragFiles Parameter - hptrDrag

hptrDrag ([HPOINTER](#)) - input
Icon to display during the drag.

DrgDragFiles Parameter - vkTerm

vkTerm ([ULONG](#)) - input
Button that ends the drag.

Possible values are described in the following list:

VK_BUTTON1 Release of button 1 ends the drag.

VK_BUTTON2 Release of button 2 ends the drag.

VK_BUTTON3 Release of button 3 ends the drag.

VK_ENDDRAG Release of the system-defined direct manipulation button ends the drag. This is the recommended value if the [DrgDrag](#) function call is invoked in response to a [WM_BEGINDRAG](#) message.

DrgDragFiles Parameter - fSourceRender

fSourceRender ([BOOL](#)) - input
Flag indicating whether the source must perform the move or copy.

TRUE The caller will receive a [DM_RENDERFILE](#) message for each file.

FALSE All file manipulation is performed by DrgDragFiles.

DrgDragFiles Parameter - ulReserved

ulReserved ([ULONG](#)) - input
Reserved.

DrgDragFiles Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	The drag operation was initiated successfully.
FALSE	An error occurred.

DrgDragFiles - Parameters

Hwnd ([HWND](#)) - input
Handle of calling window.

pFiles ([PAPSZ](#)) - input
The names of the files to be dragged.

pTypes ([PAPSZ](#)) - input
The file types of the files to be dragged.

pTargets ([PAPSZ](#)) - input
Target file names.

cFiles ([ULONG](#)) - input
Number of files to be dragged.

hptrDrag ([HPOINTER](#)) - input
Icon to display during the drag.

vkTerm ([ULONG](#)) - input
Button that ends the drag.

Possible values are described in the following list:

VK_BUTTON1	Release of button 1 ends the drag.
VK_BUTTON2	Release of button 2 ends the drag.
VK_BUTTON3	Release of button 3 ends the drag.
VK_ENDDRAG	Release of the system-defined direct manipulation button ends the drag. This is the recommended value if the DrgDrag function call is invoked in response to a WM_BEGINDRAG message.

fSourceRender ([BOOL](#)) - input
Flag indicating whether the source must perform the move or copy.

TRUE	The caller will receive a DM_RENDERFILE message for each file.
FALSE	All file manipulation is performed by DrgDragFiles.

ulReserved ([ULONG](#)) - input
Reserved.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	The drag operation was initiated successfully.
FALSE	An error occurred.

DrgDragFiles - Remarks

This function begins a direct manipulation operation for one or more files. [DRAGINFO](#) and [DRAGITEM](#) structures are allocated and initialized, and are then used as input to [DrgDrag](#). All of the post-drag conversation required to complete the direct manipulation operation is handled by an object window created by this function.

The caller should set *fSourceRender* to TRUE if it must perform the file manipulation for any of these files. When *fSourceRender* is TRUE, the caller receives a [DM_RENDERFILE](#) message as the drag-object window receives a [DM_RENDER](#) message. The caller should move or copy the file after receiving the [DM_RENDERFILE](#) message. The caller should then send a [DM_FILERENDERED](#) message to the drag-object window, and the drag-object window should send a [DM_RENDERERCOMPLETE](#) message to the target.

When *pTypes* is NULL, the .TYPE EA is interrogated to determine the type for each file in *pFiles*. When *pTypes* is not NULL, the size of the array is expected to be the same as the size of *pFiles*. When any individual pointer in the array is NULL, the .TYPE EA for the corresponding file is read. When .TYPE EA does not exist for any file for which it is needed, a type of DRT_UNKNOWN is used.

When *pTargets* is NULL, the target name for a file will be the same as the source file name with the path information removed. If *pTargets* is not NULL, the size of the array is expected to be the same as the size of *pFiles*. If any individual pointer in the array is NULL, the target name for the corresponding file will match the source name minus the path information.

The rendering mechanism and format for each file is:

```
<DRM_OS2FILE,DRF_UNKNOWN> .
```

When an error occurs during the move or copy, the caller is sent a [DM_DRAGERROR](#) message. The caller can take corrective action.

As the operation is complete for each file in the list, a [DM_DRAGFILECOMPLETE](#) message is sent to the caller of DrgDragFiles. The caller is thus notified that resources can be freed for a particular file.

This function returns TRUE if the drag operation was initiated successfully and FALSE if an error occurred.

DrgDragFiles - Related Functions

Related Functions

- [DrgAcceptDroppedFiles](#)
-

DrgDragFiles - Example Code

This example calls the DrgDragFiles function to begin direct manipulation for a single file object, using the same source and target name, and determining the file type based on the file's type EA.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#define INCL_WININPUT  /* Window Input Functions */
#include <os2.h>

BOOL      fSuccess;      /* Indicate success or failure */
HWND      Hwnd;          /* Handle of calling window */
PSZ       pFiles[1];     /* The names of the files to be dragged */
PSZ       pTypes[1];     /* The file types of the files to be
                          /* dragged
PSZ       pTargets[1];   /* The target file names
HPOINTER  hptrDrag;      /* Icon to display during drag

pFiles[0] = "FILENAME.EXT"; /* Copy file name to string array */
```

```
pTargets[0] = NULL;          /* Use source name as target name */
pTypes[0] = NULL;           /* Query type EA to determine file type */

fSuccess = DrgDragFiles(Hwnd, pFiles, pTypes, pTargets, 1,
                        hptrDrag, VK_BUTTON2, FALSE, 0L);
```

DrgDragFiles - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgFreeDraginfo

DrgFreeDraginfo - Syntax

This function frees a [DRAGINFO](#) structure allocated by [DrgAllocDraginfo](#).

```
#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGINFO    pdinfo; /* Pointer to the DRAGINFO structure. */
BOOL         rc;     /* Success indicator. */

rc = DrgFreeDraginfo(pdinfo);
```

DrgFreeDraginfo Parameter - pdinfo

pdinfo ([PDRAGINFO](#)) - input
Pointer to the [DRAGINFO](#) structure.

DrgFreeDraginfo Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgFreeDraginfo - Parameters

pdinfo ([PDRAGINFO](#)) - input
Pointer to the [DRAGINFO](#) structure.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgFreeDraginfo - Remarks

DrgFreeDraginfo fails with an error of PMERR_SOURCE_SAME_AS_TARGET if it is called by the process that called [DrgDrag](#) before [DrgDrag](#) returns. When a process is performing a drag operation between two of its own windows, this prevents the source window from freeing the [DRAGINFO](#) structure before the target window finishes processing.

DrgFreeDraginfo - Errors

Possible returns from [WinGetLastError](#)

PMERR_MEMORY_DEALLOCATION_ERR (0x1113)
An error occurred during memory management.

PMERR_SOURCE_SAME_AS_TARGET (0x1502)
The direct manipulation source and target process are the same.

DrgFreeDraginfo - Related Functions

Prerequisite Functions

- [DrgAllocDraginfo](#)

Related Functions

- [DrgAccessDraginfo](#)
- [DrgDrag](#)
- [DrgPushDraginfo](#)

DrgFreeDraginfo - Example Code

This example calls the DrgFreeDraginfo function to free an existing [DRAGINFO](#) structure allocated by the [DrgAllocDraginfo](#) function after a drag operation has completed.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

BOOL      fSuccess;      /* Indicate success or failure          */
PDRAGINFO pdinfo;        /* Pointer to DRAGINFO structure          */
HWND      hwnd;          /* Handle of calling (source) window      */
DRAGIMAGE dimg;          /* DRAGIMAGE structure                    */
HWND      hwndDrop;      /* Handle of drop (target) window         */

/*****
/* Perform the drag operation:
/* - Give the user a visual cue by changing the pointer to a
/*   bit map
/* - Send DM_DRAGOVER messages to the target window (in this
/*   case it is also the source)
*****/
hwndDrop = DrgDrag(hwnd,          /* Source of the drag          */
                  pdinfo,        /* Pointer to DRAGINFO structure */
                  (PDRAGIMAGE)&dimg, /* Drag image                  */
                  1,             /* Size of the pdimg array     */
                  VK_ENDDRAG,    /* Release of drag button      */
                  /* Terminates the drag          */
                  NULL);         /* Reserved                     */

fSuccess = DrgFreeDraginfo(&pdinfo);
```

DrgFreeDraginfo - Topics

Select an item:

- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Errors](#)
 - [Remarks](#)
 - [Example Code](#)
 - [Related Functions](#)
 - [Glossary](#)
-

DrgFreeDragtransfer

DrgFreeDragtransfer - Syntax

This function frees the storage associated with a [DRAGTRANSFER](#) structure.

```
#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGTRANSFER    pdxfer; /* Pointer to the DRAGTRANSFER structures to be freed. */
BOOL              rc;     /* Return code. */

rc = DrgFreeDragtransfer(pdxfer);
```

DrgFreeDragtransfer Parameter - pdxfer

pdxfer ([PDRAGTRANSFER](#)) - input
Pointer to the [DRAGTRANSFER](#) structures to be freed.

DrgFreeDragtransfer Return Value - rc

rc ([BOOL](#)) - returns
Return code.

TRUE	The structure was freed successfully.
FALSE	The deallocation failed.

DrgFreeDragtransfer - Parameters

pdxfer ([PDRAGTRANSFER](#)) - input
Pointer to the [DRAGTRANSFER](#) structures to be freed.

rc ([BOOL](#)) - returns
Return code.

TRUE	The structure was freed successfully.
FALSE	The deallocation failed.

DrgFreeDragtransfer - Remarks

This function frees the [DRAGTRANSFER](#) structures allocated by calls to [DrgAllocDragtransfer](#). When all of the [DRAGTRANSFER](#)

structures have been freed, the memory block containing the [DRAGTRANSFER](#) array is deallocated.

DrgFreeDragtransfer - Errors

Possible returns from [WinGetLastError](#)

PMERR_MEMORY_DEALLOCATION_ERR (0x1113)
An error occurred during memory management.

DrgFreeDragtransfer - Related Functions

Related Functions

- [DrgAllocDragtransfer](#)
-

DrgFreeDragtransfer - Example Code

This example calls the DrgFreeDragtransfer function to free an existing [DRAGTRANSFER](#) structure allocated by the [DrgAllocDragtransfer](#) function.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

BOOL          fSuccess; /* Indicate success or failure          */
DRAGTRANSFER dxfer;     /* Pointer to DRAGTRANSFER structure */

fSuccess = DrgFreeDragtransfer(&dxfer);
```

DrgFreeDragtransfer - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgGetPS

DrgGetPS - Syntax

This function gets a presentation space that is used to provide target feedback to the user during a drag operation.

```
#define INCL_WINSTDDRAG
#include <os2.h>

HWND    hwnd; /* Handle of the window for which presentation space is required. */
HPS     Hps;  /* Presentation-space handle used for drawing in the window. */

Hps = DrgGetPS(hwnd);
```

DrgGetPS Parameter - hwnd

hwnd (**HWND**) - input
Handle of the window for which presentation space is required.

DrgGetPS Return Value - Hps

Hps (**HPS**) - returns
Presentation-space handle used for drawing in the window.

NULLHANDLE
Error occurred.

DrgGetPS - Parameters

hwnd (**HWND**) - input
Handle of the window for which presentation space is required.

Hps (**HPS**) - returns
Presentation-space handle used for drawing in the window.

NULLHANDLE
Error occurred.

DrgGetPS - Remarks

This function returns a handle to a presentation space that can be used for drawing while a direct manipulation operation is in progress.

DrgGetPS is called only during a direct manipulation operation. This function is called only after a [DM_DRAGOVER](#), [DM_DRAGLEAVE](#), or [DM_DROP](#) message has been received.

In order to draw target emphasis, an application must use DrgGetPS and [DrgReleasePS](#) to unlock its window.

The presentation space created with DrgGetPS must be freed with [DrgReleasePS](#).

DrgGetPS - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_NOT_DRAGGING (0x1f00)
A drag operation is not in progress at this time.

DrgGetPS - Related Functions

Related Functions

- [DrgReleasePS](#)

DrgGetPS - Example Code

This example uses the DrgGetPS function to get a presentation space handle which is used during drag operations such as loading a drag bit map. When finished with the presentation space, release it with the [DrgReleasePS](#) function.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

HPS    hps;                /* Presentation space handle */
HWND   hwnd;               /* Handle of the window for which
                           /* presentation space is required */

case DM_DRAGOVER:
    hps = DrgGetPS(hwnd);

DrawTargetEmphasis(hps, hwnd);
DrgReleasePS(hps);
```

DrgGetPS - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)

DrgLazyDrag

DrgLazyDrag - Syntax

DrgLazyDrag is called when a direct-manipulation button is pressed while the lazy drag augmentation key is held to initiate a pickup and drop (lazy drag) operation.

```
#define INCL_WINSTDDRAG
#include <os2.h>

HWND      hwndSource; /* Handle of the source window that is calling this function. */
PDRAGINFO pdinfo;     /* Pointer to the DRAGINFO structure which contains information about the objects being dragged. */
PDRAGIMAGE pdimg;      /* Pointer to an array of DRAGIMAGE structures. */
ULONG      cdimg;       /* Number of DRAGIMAGE structures in the pdimg array. */
PVOID      pRsvd;       /* Reserved value, must be 0. */
BOOL       rc;          /* Success indicator. */

rc = DrgLazyDrag(hwndSource, pdinfo, pdimg,
                  cdimg, pRsvd);
```

DrgLazyDrag Parameter - hwndSource

hwndSource ([HWND](#)) - input
Handle of the source window that is calling this function.

DrgLazyDrag Parameter - pdinfo

pdinfo ([PDRAGINFO](#)) - input
Pointer to the [DRAGINFO](#) structure which contains information about the objects being dragged.

DrgLazyDrag Parameter - pdimg

pdimg ([PDRAGIMAGE](#)) - input
Pointer to an array of [DRAGIMAGE](#) structures.

These structures describe the images that are to be drawn under the direct-manipulation pointer during the drag.

DrgLazyDrag Parameter - cdimg

cdimg ([ULONG](#)) - input
Number of [DRAGIMAGE](#) structures in the *pdimg* array.

DrgLazyDrag Parameter - pRsvd

pRsvd ([PVOID](#)) - input
Reserved value, must be 0.

DrgLazyDrag Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

Possible values are described in the following list:

- | | |
|-------|---|
| TRUE | A lazy drag operation was successfully started. |
| FALSE | An error occurred while initiating a lazy drag operation. |

DrgLazyDrag - Parameters

hwndSource ([HWND](#)) - input
Handle of the source window that is calling this function.

pdinfo ([PDRAGINFO](#)) - input
Pointer to the [DRAGINFO](#) structure which contains information about the objects being dragged.

pdimg ([PDRAGIMAGE](#)) - input
Pointer to an array of [DRAGIMAGE](#) structures.

These structures describe the images that are to be drawn under the direct-manipulation pointer during the drag.

cdimg ([ULONG](#)) - input
Number of [DRAGIMAGE](#) structures in the *pdimg* array.

pRsvd ([PVOID](#)) - input

Reserved value, must be 0.

rc (**BOOL**) - returns
Success indicator.

Possible values are described in the following list:

TRUE	A lazy drag operation was successfully started.
FALSE	An error occurred while initiating a lazy drag operation.

DrgLazyDrag - Remarks

Before initiating a lazy drag operation by calling `DrgLazyDrag`, the application must allocate the **DRAGINFO** structure using `DrgAllocDraginfo`.

`DrgLazyDrag` is called when the direct-manipulation button is pressed while holding down the lazy drag augmentation key, currently the ALT key. As the pointer moves over a potential target, a **DM_DRAGOVER** message is sent to the target window. When the pointer moves from one target window to another, a **DM_DRAGLEAVE** message is sent to the former target.

If the pointer is over a valid window target when the direct-manipulation button is pressed and the lazy drag "end drag" augmentation key (Ctrl+Shift to link, Ctrl to copy and Shift to move) is pressed, a **DM_DROP** message is sent to the target window. The source window posts a **DM_DROPNOTIFY** message, informing the target window that a drop has occurred.

param2 parameter of **DM_DROPNOTIFY** message contains the handle of the target window that the drag set was dropped on. If *param2* is zero, the drag set was not dropped; in other words, the drag operation was canceled. The source window must check *pDraginfo* to see if the target window is different from the source. If the source and target are different, the source window must free **DRAGINFO** upon receipt of the **DM_DROPNOTIFY** message. If the source and target window handles are the same, the target must free the **DRAGINFO** after completing the post-drop conversation.

The caller can define a default operation for the objects represented by the **DRAGINFO** structure by modifying its *usOperate* field. If *usOperate* is modified, the new value is sent to the target as the operation when a NO_DEFAULT operation would normally be sent. The caller must not modify any other part of the **DRAGINFO** structure. The **DRAGITEM** structures associated with the **DRAGINFO** structure must only be altered with `DrgSetDragitem` or by using a pointer obtained from `DrgQueryDragitemPtr`.

A window receives a **WM_PICKUP** message when the direct-manipulation button is pressed holding down the lazy drag augmentation key. In response to this message, an application is responsible for:

- Obtaining a **DRAGINFO** structure using `DrgAllocDraginfo`
- Initializing the **DRAGITEM** structures using `DrgSetDragitem`.

Objects are added to the drag set whenever a **WM_PICKUP** message is received. The first time that message is received, the application must call `DrgLazyDrag` to begin the lazy drag operation. Each subsequent **WM_PICKUP** message that is received during the course of a lazy drag operation indicates that objects are to be added to the drag set. If objects are currently being dragged, the application must reallocate the **DRAGINFO** structure using `DrgReallocDragInfo`. `DrgReallocDragInfo` frees the current **DRAGINFO** and allocates a new one. The lazy drag operation can then continue by making another call to `DrgLazyDrag`.

Objects can also be removed from the drag set during the course of a lazy drag operation. It is the application's responsibility to define what action initiates such a "putback" operation; for example, if a "pickup" operation executed while the pointer is not over a valid object. It is the application's responsibility to decide if a putback operation may apply to individual objects within the drag set or the drag set as a whole. If the putback operation is applied to the drag set as a whole, the result is the same as canceling the drag. If the put-back operation is applied to individual objects within the drag set, the application must free the **DRAGINFO** and reallocate it to represent the new state of the drag set. If the drag operation is to continue, the application must make another call to `DrgLazyDrag`.

`DrgLazyDrag` returns as soon as it completes initialization for the drag. The pointing device remains active during a lazy drag and can be used in the same manner as if no drag operation were in progress. As soon as `DrgLazyDrag` returns, the application can free its **DRAGIMAGE** array.

Note: Because the lazy drag operation is nonmodal, the mouse pointer may be used as if no drag operation were in progress.

Once a standard drag operation is commenced by calling `DrgDrag`, the objects that comprise the drag set cannot be modified without canceling the drag and restarting. The drag set can be modified during the course of the drag operation. When the drag set is modified, the application must reallocate the **DRAGINFO** structure and make another call to `DrgLazyDrag`.

Note: The mouse changes to an arrow with an attache case attached to the lower right-hand corner when the lazy drag is in progress, rather

than a stack of images shown during a standard drag.

The *pdimg* and *cdimg* parameters were added for compatibility with [DrgDrag](#) only.

DrgLazyDrag - Related Functions

Prerequisite Functions

- [DrgAllocDraginfo](#)

Related Functions

- [DrgDrag](#)
 - [DrgFreeDraginfo](#)
 - [DrgQueryDragitemPtr](#)
 - [DrgSetDragitem](#)
-

DrgLazyDrag - Example Code

This example shows the proper sequence for initiating a lazy drag operation after the user has selected an object and pressed the direct manipulation button while holding down the lazy drag augmentation key (ALT). The window receives a [WM_PICKUP](#) message indicating that a lazy drag operation is to begin.

```
#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGINFO pdinfo; /* Pointer to a DRAGINFO structure */
HWND hwndSource; /* Handle of the Source window */
DRAGITEM ditem; /* DRAGITEM structure */
DRAGIMAGE pdimg; /* Pointer to DRAGIMAGE structure */
HBITMAP hbm; /* Bit-map handle passed to DrgLazyDrag */
.
.
.
case WM_PICKUP:

    /******
    /* Initialize the DRAGITEM structure */
    /******
    ditem.hwndItem=hwndSource; /* Handle of the source window */
    ditem.ulItemID=ID_ITEM; /* App defined id of item */
    ditem.hstrType=DrgAddStrHandle("DRT_TEXT"); /* Text item */
    ditem.hstrRMF=DrgAddStrHandle("<DRM_OS2FILE,DRF_TEXT>");
    ditem.hstrContainerName=DrgAddStrHandle("C:\\");
    ditem.hstrSourceName=DrgAddStrHandle("C:\\\\CONFIG.SYS");
    ditem.hstrTargetName=DrgAddStrHandle("C:\\OS2\\CONFIG.SYS");
    ditem.cxOffset=0; /* X-offset of the origin of the image from the */
    /* pointer hotspot */
    ditem.cyOffset=0; /* Y-offset of the origin of the image from the */
    /* pointer hotspot */
    ditem.fsControl=0; /* Source item control flags */
    ditem.fsSupportedOps=0;

    /******
    /* Create the DRAGINFO structure */
    /******
    pdinfo=DrgAllocDraginfo(1);
    if(!pdinfo) return FALSE; /* Return FALSE if initialization fails */

    /******
    /* Initialize the DRAGIMAGE structure */
    /*
```

```

/*****
pdimg=AllocMem(sizeof(DRAGIMAGE));

pdimg->cb=sizeof(DRAGIMAGE);      /* Size of the dragimage structure */
pdimg->cptl=0;                     /* Image is not a polygon */
pdimg->hImage=hbm;                /* Handle of image to display */
pdimg->szlStretch.cx=20L          /* Size to stretch icon or bit map */
pdimg->fl=DRG_BITMAP|DRG_STRETCH; /* Flags passed to DrgLazyDrag */
pdimg->cxOffset=0;                /* Offset of the origin of image */
pdimg->cyOffset=0;                /* from the pointer hotspot */

/*****
/* Set the DRAGITEM */
/*****
DrgSetDragitem(pdinfo, &ditem, (ULONG)sizeof(ditem, 0);
/*****
/* Begin the Lazy Drag operation */
/*****
if (DrgLazyDrag (hwndSource,      /* Source of the drag */
                pdinfo,          /* Pointer to the DRAGINFO */
                pdimg,           /* DRAGIMAGE array */
                1,               /* Size of the DRAGIMAGE array */
                NULL)) {         /* Reserved */
    FreeMem (pdimg);             /* Free DRAGIMAGE if successful */
}

```

DrgLazyDrag - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgLazyDrop

DrgLazyDrop - Syntax

This function is called to invoke a lazy drop operation.

```

#define INCL_WINSTDDRAG
#include <os2.h>

HWND      hwndTarget; /* Handle of the target window receiving the drop. */
ULONG     ulOperation; /* Drop operation code. */
PPOINTL   pptlDrop;    /* Pointer to the drop location in desktop coordinates. */
BOOL      rc;          /* Success indicator. */

rc = DrgLazyDrop(hwndTarget, ulOperation,
                 pptlDrop);

```

DrgLazyDrop Parameter - hwndTarget

hwndTarget (HWND) - input
Handle of the target window receiving the drop.

DrgLazyDrop Parameter - ulOperation

ulOperation (ULONG) - input
Drop operation code.

Possible values are shown in the following list:

DO_DEFAULT	Default operation.
DO_COPY	Operation is a copy.
DO_MOVE	Operation is a move.
DO_LINK	Operation is a link.

DrgLazyDrop Parameter - pptIDrop

pptIDrop (PPOINTL) - input
Pointer to the drop location in desktop coordinates.

DrgLazyDrop Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE	Objects are successfully dropped.	.
FALSE	An error occurred.	

DrgLazyDrop - Parameters

hwndTarget (**HWND**) - input
Handle of the target window receiving the drop.

ulOperation (**ULONG**) - input
Drop operation code.

Possible values are shown in the following list:

- DO_DEFAULT** Default operation.
- DO_COPY** Operation is a copy.
- DO_MOVE** Operation is a move.
- DO_LINK** Operation is a link.

ptlDrop (**POINTL**) - input
Pointer to the drop location in desktop coordinates.

rc (**BOOL**) - returns
Success indicator.

- TRUE** Objects are successfully dropped.
- FALSE** An error occurred.

DrgLazyDrop - Remarks

This function can be used to implement a "drop" choice from a menu.

DrgLazyDrop - Example Code

This example uses DrgLazyDrop to perform a drop to a target window, using the copy operation.

```
#define INCL_WINSTDDRAG
#include <os2.h>

HWND      hwndTarget;    /* Handle of the Target window */
ULONG     ulDropOp;      /* Operation to be performed on the drop */
POINTL    ptlDrop;       /* Drop location */
BOOL      bSuccess;      /* Return code from API */

ptlDrop.x=5;
ptlDrop.y=10;

usDropOp=DO_COPY;
bSuccess=DrgLazyDrop(hwndTarget, ulDropOp, &ptlDrop);
```

DrgLazyDrop - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

DrgPostTransferMsg

DrgPostTransferMsg - Syntax

This function posts a message to the other application involved in the direct manipulation operation.

```
#define INCL_WINSTDDRAG
#include <os2.h>

HWND      hwnd;      /* Window handle to which the message      is to be posted. */
ULONG     msg;       /* Identifier of the message to be posted. */
PDRAGTRANSFER pdxfer; /* Pointer to the DRAGTRANSFER structure. */
ULONG     fl;        /* Flags to be passed in the param2 parameter of the message identified by msg. */
ULONG     ulRsvd;    /* Reserved value, must be 0. */
BOOL      fRetry;    /* Retry indicator. */
BOOL      rc;        /* Success indicator. */

rc = DrgPostTransferMsg(hwnd, msg, pdxfer,
                        fl, ulRsvd, fRetry);
```

DrgPostTransferMsg Parameter - hwnd

hwnd (**HWND**) - input
Window handle to which the message is to be posted.

Target
hwndItem in the **DRAGITEM** structure.

Source
hwndClient in the **DRAGTRANSFER** structure.

DrgPostTransferMsg Parameter - msg

msg (**ULONG**) - input
Identifier of the message to be posted.

DM_RENDERCOMPLETE is the only valid message.

DrgPostTransferMsg Parameter - pdxfer

pdxfer ([PDRAGTRANSFER](#)) - input
Pointer to the [DRAGTRANSFER](#) structure.

DrgPostTransferMsg Parameter - fl

fl ([ULONG](#)) - input
Flags to be passed in the *param2* parameter of the message identified by *msg*.

DrgPostTransferMsg Parameter - ulRsvd

ulRsvd ([ULONG](#)) - input
Reserved value, must be 0.

DrgPostTransferMsg Parameter - fRetry

fRetry ([BOOL](#)) - input
Retry indicator.

TRUE	<p>If the destination queue is full, the message posting is retried at 1-second intervals until the message is posted successfully.</p> <p>In this case, DrgPostTransferMsg dispatches any messages in the queue by calling WinPeekMsg and WinDispatchMsg in a loop. The application can receive messages sent by other applications while it is trying to post drag transfer messages.</p>
FALSE	<p>The call returns FALSE without retrying.</p>

DrgPostTransferMsg Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE

FALSE	Successful completion.
	Error occurred.

DrgPostTransferMsg - Parameters

hwnd ([HWND](#)) - input
Window handle to which the message is to be posted.

Target
hwndItem in the [DRAGITEM](#) structure.

Source
hwndClient in the [DRAGTRANSFER](#) structure.

msg ([ULONG](#)) - input
Identifier of the message to be posted.

[DM_RENDERCOMPLETE](#) is the only valid message.

pdxfer ([PDRAGTRANSFER](#)) - input
Pointer to the [DRAGTRANSFER](#) structure.

fl ([ULONG](#)) - input
Flags to be passed in the *param2* parameter of the message identified by *msg*.

ulRsvd ([ULONG](#)) - input
Reserved value, must be 0.

fRetry ([BOOL](#)) - input
Retry indicator.

TRUE
If the destination queue is full, the message posting is retried at 1-second intervals until the message is posted successfully.

In this case, DrgPostTransferMsg dispatches any messages in the queue by calling [WinPeekMsg](#) and [WinDispatchMsg](#) in a loop. The application can receive messages sent by other applications while it is trying to post drag transfer messages.

FALSE
The call returns FALSE without retrying.

rc ([BOOL](#)) - returns
Success indicator.

TRUE
Successful completion.

FALSE
Error occurred.

DrgPostTransferMsg - Remarks

The *fsReply* field in the [DRAGTRANSFER](#) structure is set to 0 before the message is posted. If the posting fails for any reason, FALSE is returned.

DrgPostTransferMsg - Related Functions

Related Functions

- [DrgSendTransferMsg](#)

DrgPostTransferMsg - Example Code

This example calls the `DrgPostTransferMsg` function to respond to a [DM_RENDER](#) message from the target. The response consists of a [DM_RENDERCOMPLETE](#) message, plus a flag indicating whether the render was successful (`DMFL_RENDEROK`) or not (`DMFL_RENDERFAIL`).

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

MPARAM      mpl;          /* Message parameter 1 */
BOOL        fSuccess;     /* Indicate success or failure */
BOOL        Rendered;     /* Success of render operation */
PDRAGTRANSFER pdxfer;     /* Pointer to DRAGTRANSFER structure */

case DM_RENDER:
    pdxfer = (PDRAGTRANSFER)PVOIDFROMMP(mpl); /* Get DRAGTRANSFER */
                                              /* structure */

    /* Attempt to render file */

    if (Rendered)
    {
        fSuccess = DrgPostTransferMsg(pdxfer->pditem,
                                      DM_RENDERCOMPLETE,
                                      pdxfer,
                                      DMFL_RENDEROK,
                                      0, FALSE);

        return (MRESULT)TRUE;
    }
    else
    {
        fSuccess = DrgPostTransferMsg(pdxfer->pditem,
                                      DM_RENDERCOMPLETE,
                                      pdxfer,
                                      DMFL_RENDERFAIL,
                                      0, FALSE);

        return (MRESULT)FALSE;
    }
}
```

DrgPostTransferMsg - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgPushDraginfo

DrgPushDraginfo - Syntax

This function gives a process access to a [DRAGINFO](#) structure.

```
#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGINFO    pdinfo;    /* Pointer to the DRAGINFO structure. */
HWND         hwndDest;  /* Handle of the window whose process is to be given access to a DRAGINFO structure. */
BOOL         rc;        /* Success indicator. */

rc = DrgPushDraginfo(pdinfo, hwndDest);
```

DrgPushDraginfo Parameter - pdinfo

pdinfo ([PDRAGINFO](#)) - input
Pointer to the [DRAGINFO](#) structure.

DrgPushDraginfo Parameter - hwndDest

hwndDest ([HWND](#)) - input
Handle of the window whose process is to be given access to a [DRAGINFO](#) structure.

DrgPushDraginfo Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgPushDraginfo - Parameters

pdinfo ([PDRAGINFO](#)) - input
Pointer to the [DRAGINFO](#) structure.

hwndDest ([HWND](#)) - input
Handle of the window whose process is to be given access to a [DRAGINFO](#) structure.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgPushDraginfo - Remarks

The receiving process is responsible for:

1. Deleting the string handles in the [DRAGINFO](#) structure with [DrgDeleteDraginfoStrHandles](#)
2. Freeing the [DRAGINFO](#) structure using [DrgFreeDraginfo](#).

DrgPushDraginfo - Errors

Possible returns from [WinGetLastError](#)

PMERR_ACCESS_DENIED (0x150D)
The memory block was not allocated properly.

PMERR_INSUFFICIENT_MEMORY (0x203E)
The operation terminated through insufficient memory.

DrgPushDraginfo - Related Functions

Related Functions

- [DrgAccessDraginfo](#)
- [DrgAllocDraginfo](#)
- [DrgDrag](#)
- [DrgFreeDraginfo](#)

DrgPushDraginfo - Example Code

This example calls the [DrgPushDraginfo](#) function to grant access to a [DRAGINFO](#) structure to the process owning the specified window handle. The [DRAGINFO](#) structure was previously allocated using the [DrgAllocDraginfo](#) function.

```

#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

BOOL      fSuccess;      /* Indicate success or failure */
DRAGINFO  Draginfo;      /* Pointer to DRAGINFO structure */
HWND      hwndDest;      /* Handle of window whose process will */
                        /* will be given access to the DRAGINFO */
                        /* structure */

fSuccess = DrgPushDraginfo(&Draginfo,hwndDest);

```

DrgPushDraginfo - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgQueryDraginfoPtr

DrgQueryDraginfoPtr - Syntax

This function obtains a pointer to the current [DRAGINFO](#) structure.

```

#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGINFO  pRsvd;      /* Reserved value, must be NULL. */
PDRAGINFO  pDragInfo;  /* Pointer to the current DRAGINFO structure. */

pDragInfo = DrgQueryDraginfoPtr(pRsvd);

```

DrgQueryDraginfoPtr Parameter - pRsvd

pRsvd ([PDRAGINFO](#)) - input
 Reserved value, must be NULL.

DrgQueryDraginfoPtr Return Value - pDragInfo

pDragInfo ([PDRAGINFO](#)) - returns
Pointer to the current [DRAGINFO](#) structure.

A return value of NULL indicates that a [DRAGINFO](#) has not been allocated.

DrgQueryDraginfoPtr - Parameters

pRsvd ([PDRAGINFO](#)) - input
Reserved value, must be NULL.

pDragInfo ([PDRAGINFO](#)) - returns
Pointer to the current [DRAGINFO](#) structure.

A return value of NULL indicates that a [DRAGINFO](#) has not been allocated.

DrgQueryDraginfoPtr - Remarks

The returned [DRAGINFO](#) structure could have been allocated for use in either a standard or lazy drag operation. [DrgQueryDragStatus](#) can be used to determine which type of drag is active.

DrgQueryDraginfoPtr - Related Functions

Related Functions

- [DrgQueryDraginfoPtrFromDragitem](#)
 - [DrgQueryDraginfoPtrFromHwnd](#)
-

DrgQueryDraginfoPtr - Example Code

This example uses DrgQueryDraginfoPtr to obtain a pointer to the current [DRAGINFO](#) structure.

```
#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGINFO pdinfo;          /* Pointer to a DRAGINFO structure */

pdinfo = DrgQueryDraginfoPtr(NULL);
if (pdinfo)
{
    .
    .
}
```

```
} .
```

DrgQueryDraginfoPtr - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

DrgQueryDraginfoPtrFromDragitem

DrgQueryDraginfoPtrFromDragitem - Syntax

This function is called to obtain a pointer to the [DRAGINFO](#) structure associated with a given [DRAGITEM](#) structure.

```
#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGITEM    pditem;      /* Pointer to a DRAGITEM structure whose corresponding DRAGINFO is to be returned. */
PDRAGINFO    pDraginfo;   /* Pointer to the DRAGINFO structure for the specified pditem. */

pDraginfo = DrgQueryDraginfoPtrFromDragitem(
    pditem);
```

DrgQueryDraginfoPtrFromDragitem Parameter - pditem

pditem ([PDRAGITEM](#)) - input

Pointer to a [DRAGITEM](#) structure whose corresponding [DRAGINFO](#) is to be returned.

DrgQueryDraginfoPtrFromDragitem Return Value - pDraginfo

pDraginfo ([PDRAGINFO](#)) - returns
Pointer to the [DRAGINFO](#) structure for the specified *pditem*.

A return value of NULL indicates that a [DRAGITEM](#) structure was not found.

DrgQueryDraginfoPtrFromDragitem - Parameters

pditem ([PDRAGITEM](#)) - input
Pointer to a [DRAGITEM](#) structure whose corresponding [DRAGINFO](#) is to be returned.

pDraginfo ([PDRAGINFO](#)) - returns
Pointer to the [DRAGINFO](#) structure for the specified *pditem*.

A return value of NULL indicates that a [DRAGITEM](#) structure was not found.

DrgQueryDraginfoPtrFromDragitem - Example Code

This example uses DrgQueryDraginfoPtrFromDragitem to obtain the pointer to the [DRAGINFO](#) structure associated with a given [DRAGITEM](#) structure. The example assumes the application already has a pointer to one of the [DRAGITEM](#) structures.

```
#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGINFO pdinfo;          /* Pointer to a DRAGINFO structure */
PDRAGITEM pdragitem;        /* Pointer to a DRAGITEM structure */

pdinfo=DrgQueryDraginfoPtrFromDragitem(pdragitem);
if (pdinfo) {
    .
    .
    .
}
```

DrgQueryDraginfoPtrFromDragitem - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Example Code](#)
 - [Glossary](#)

DrgQueryDraginfoPtrFromHwnd

DrgQueryDraginfoPtrFromHwnd - Syntax

This function determines whether a particular window has allocated a [DRAGINFO](#) structure.

```
#define INCL_WINSTDDRAG
#include <os2.h>

HWND      hwndSource; /* Handle of the window whose associated DRAGINFO pointer is to be returned.
PDRAGINFO  pDraginfo; /* Pointer to the DRAGINFO structure allocated by the window specified by hwndSource.

pDraginfo = DrgQueryDraginfoPtrFromHwnd(hwndSource);
```

DrgQueryDraginfoPtrFromHwnd Parameter - hwndSource

hwndSource ([HWND](#)) - input
Handle of the window whose associated [DRAGINFO](#) pointer is to be returned.

DrgQueryDraginfoPtrFromHwnd Return Value - pDraginfo

pDraginfo ([PDRAGINFO](#)) - returns
Pointer to the [DRAGINFO](#) structure allocated by the window specified by *hwndSource*.
If the return value is NULL, the [DRAGINFO](#) structure has not been allocated.

DrgQueryDraginfoPtrFromHwnd - Parameters

hwndSource ([HWND](#)) - input
Handle of the window whose associated [DRAGINFO](#) pointer is to be returned.

pDraginfo ([PDRAGINFO](#)) - returns
Pointer to the [DRAGINFO](#) structure allocated by the window specified by *hwndSource*.
If the return value is NULL, the [DRAGINFO](#) structure has not been allocated.

DrgQueryDraginfoPtrFromHwnd - Remarks

The application can gain access to the structure [DRAGINFO](#) by calling [DrgAccessDraginfo](#).

DrgQueryDraginfoPtrFromHwnd - Example Code

This example uses DrgQueryDraginfoPtrFromHwnd to obtain a pointer to the [DRAGINFO](#) structure allocated by the given window.

```
#define INCL_WINSTDDRAG
#include <os2.h>

HWND      hwndWindow;    /* Window handle */
PDRAGINFO pinfo;         /* Pointer to a DRAGINFO structure */

pdinfo=DrgQueryDraginfoPtrFromHwnd(hwndWindow);
If (pdinfo) {
    .
    .
    .
}
```

DrgQueryDraginfoPtrFromHwnd - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

DrgQueryDragitem

DrgQueryDragitem - Syntax

This function returns a [DRAGITEM](#) structure used in the direct manipulation operation.

```
#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGINFO    pdinfo;    /* Pointer to the DRAGINFO structure from which the DRAGITEM structure is obtained. */
ULONG        cbBuffer;  /* Maximum number of bytes to copy to the buffer. */
PDRAGITEM    pditem;    /* Pointer to the buffer into which the DRAGITEM structure is copied. */
ULONG        iItem;     /* Zero-based index of the DRAGITEM to be returned. */
BOOL         rc;        /* Success indicator. */

rc = DrgQueryDragitem(pdinfo, cbBuffer, pditem,
    iItem);
```

DrgQueryDragitem Parameter - pinfo

pinfo (**PDRAGINFO**) - input
Pointer to the **DRAGINFO** structure from which the **DRAGITEM** structure is obtained.

DrgQueryDragitem Parameter - cbBuffer

cbBuffer (**ULONG**) - input
Maximum number of bytes to copy to the buffer.

DrgQueryDragitem Parameter - pditem

pditem (**PDRAGITEM**) - output
Pointer to the buffer into which the **DRAGITEM** structure is copied.

DrgQueryDragitem Parameter - item

item (**ULONG**) - input
Zero-based index of the **DRAGITEM** to be returned.

DrgQueryDragitem Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgQueryDragitem - Parameters

pdinfo ([PDRAGINFO](#)) - input
Pointer to the [DRAGINFO](#) structure from which the [DRAGITEM](#) structure is obtained.

cbBuffer ([ULONG](#)) - input
Maximum number of bytes to copy to the buffer.

pditem ([PDRAGITEM](#)) - output
Pointer to the buffer into which the [DRAGITEM](#) structure is copied.

ilitem ([ULONG](#)) - input
Zero-based index of the [DRAGITEM](#) to be returned.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgQueryDragitem - Remarks

This function returns the [DRAGITEM](#) structure identified by *ilitem*.

DrgQueryDragitem - Related Functions

Related Functions

- [DrgQueryDragitemPtr](#)
 - [DrgSetDragitem](#)
-

DrgQueryDragitem - Example Code

This example calls the `DrgQueryDragitem` function to return the entirety of the first [DRAGITEM](#) structure in the given [DRAGINFO](#) structure, after which it obtains the source window handle.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

BOOL      fSuccess;      /* Indicate success or failure */
DRAGINFO  Draginfo;      /* DRAGINFO structure from which the */
                        /* DRAGITEM structure is obtained */
ULONG     cbBuffer;      /* Maximum number of bytes to copy */
DRAGITEM  Dragitem;      /* Buffer into which the DRAGITEM */
                        /* structure is copied */
ULONG     iItem;         /* Zero-based index of the DRAGITEM */
                        /* to be returned */
HWND      hwndSource;    /* Source window handle for the drag */

cbBuffer = sizeof(DRAGITEM); /* Copy entire DRAGITEM structure */
iItem = 0;                  /* Return first DRAGITEM */

fSuccess = DrgQueryDragitem(&Draginfo,cbBuffer,&Dragitem,iItem);

hwndSource = Dragitem.hwndItem; /* Obtain source window handle */
```

DrgQueryDragitem - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

DrgQueryDragitemCount

DrgQueryDragitemCount - Syntax

This function returns the number of objects being dragged during the current direct manipulation operation.

```
#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGINFO    pdinfo; /* Pointer to the DRAGINFO structure for which number of dragged objects is requested. */
ULONG        cDitem; /* Number of objects being dragged. */

cDitem = DrgQueryDragitemCount(pdinfo);
```

DrgQueryDragitemCount Parameter - pdinfo

pdinfo ([PDRAGINFO](#)) - input

Pointer to the [DRAGINFO](#) structure for which number of dragged objects is requested.

DrgQueryDragitemCount Return Value - cDitem

cDitem ([ULONG](#)) - returns

Number of objects being dragged.

DrgQueryDragitemCount - Parameters

pdinfo ([PDRAGINFO](#)) - input
Pointer to the [DRAGINFO](#) structure for which number of dragged objects is requested.

cDitem ([ULONG](#)) - returns
Number of objects being dragged.

DrgQueryDragitemCount - Example Code

This example calls the DrgQueryDragitemCount function to return the number of [DRAGITEM](#) structures in the corresponding [DRAGINFO](#) structure, which maps to the number of objects being dragged.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

ULONG      cDitem;      /* Number of objects being dragged      */
DRAGINFO   Draginfo;    /* DRAGINFO structure queried for the    */
                        /* number of drag objects                */

cDitem = DrgQueryDragitemCount(&Draginfo);
```

DrgQueryDragitemCount - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Example Code](#)
- [Glossary](#)

DrgQueryDragitemPtr

DrgQueryDragitemPtr - Syntax

This function returns a pointer to the [DRAGITEM](#) structure used in the direct manipulation operation.

```
#define INCL_WINSTDDRAG
#include <os2.h>
```

```
PDRAGINFO    pdinfo;      /* Pointer to the DRAGINFO structure from which the DRAGITEM structure is obtained. */
ULONG        ulIndex;     /* Zero-based index of the DRAGITEM structure for which the pointer is to be returned. */
PDRAGITEM    Dragitem;     /* Pointer to the DRAGITEM structure. */
```

```
Dragitem = DrgQueryDragitemPtr(pdinfo, ulIndex);
```

DrgQueryDragitemPtr Parameter - pdinfo

pdinfo (**PDRAGINFO**) - input

Pointer to the **DRAGINFO** structure from which the **DRAGITEM** structure is obtained.

DrgQueryDragitemPtr Parameter - ulIndex

ulIndex (**ULONG**) - input

Zero-based index of the **DRAGITEM** structure for which the pointer is to be returned.

DrgQueryDragitemPtr Return Value - Dragitem

Dragitem (**PDRAGITEM**) - returns

Pointer to the **DRAGITEM** structure.

DrgQueryDragitemPtr - Parameters

pdinfo (**PDRAGINFO**) - input

Pointer to the **DRAGINFO** structure from which the **DRAGITEM** structure is obtained.

ulIndex (**ULONG**) - input

Zero-based index of the **DRAGITEM** structure for which the pointer is to be returned.

Dragitem (**PDRAGITEM**) - returns

Pointer to the **DRAGITEM** structure.

DrgQueryDragitemPtr - Remarks

This function returns a pointer to *ulItemID* in the **DRAGITEM** structure used in the direct manipulation operation.

DrgQueryDragitemPtr - Related Functions

Related Functions

- [DrgQueryDragitem](#)
-

DrgQueryDragitemPtr - Example Code

This example calls the DrgQueryDragitemPtr function to return a pointer to first [DRAGITEM](#) structure in the given [DRAGINFO](#) structure, after which it obtains the source window handle.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

PDRAGITEM  pDragitem; /* DRAGITEM structure pointer */
DRAGINFO   Draginfo; /* DRAGINFO structure from which the
                     /* DRAGITEM structure is obtained
ULONG      ulIndex; /* Zero-based index of the DRAGITEM
                     /* structure pointer to be returned
HWND       hwndSource; /* Source window handle for the drag

USHORT     usn = 0; /* Return pointer to first DRAGITEM

pDragitem = DrgQueryDragitemPtr(&Draginfo,usn);

hwndSource = pDragitem->hwndItem; /* Obtain source window handle */
```

DrgQueryDragitemPtr - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgQueryDragStatus

DrgQueryDragStatus - Syntax

This function determines the status of the current drag operation.

```
#define INCL_WINSTDDRAG
#include <os2.h>

ULONG rc; /* Flag indicating the current drag status. */

rc = DrgQueryDragStatus();
```

DrgQueryDragStatus Return Value - rc

rc (ULONG) - returns
Flag indicating the current drag status.

Possible values are shown in the following list:

0	A drag operation is not currently in progress.
DGS_DRAGINPROGRESS	A standard drag operation is in progress.
DGS_LAZYDRAGINPROGRESS	A lazy drag operation is in progress.

DrgQueryDragStatus - Parameters

rc (ULONG) - returns
Flag indicating the current drag status.

Possible values are shown in the following list:

0	A drag operation is not currently in progress.
DGS_DRAGINPROGRESS	A standard drag operation is in progress.
DGS_LAZYDRAGINPROGRESS	A lazy drag operation is in progress.

DrgQueryDragStatus - Example Code

This example uses DrgQueryDragStatus to determine whether a lazy drag operation is currently in progress.

```
#define INCL_WINSTDDRAG
#include <os2.h>

if (DrgQueryDragStatus() & DGS_LAZYDRAGINPROGRESS)
{
    .
    . /* Lazy drag is in progress */
    .
}
```

DrgQueryDragStatus - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Glossary](#)

DrgQueryNativeRMF

DrgQueryNativeRMF - Syntax

Obtains the ordered pair that represents the native rendering mechanism and format of the dragged object.

```
#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGITEM    pditem;    /* Pointer to the DRAGITEM structure. */
ULONG        cbBuffer;  /* Maximum number of bytes to copy to the buffer. */
PCHAR        pBuffer;   /* Pointer to the buffer in which the null-terminated string is to be returned. */
BOOL         rc;        /* Success indicator. */

rc = DrgQueryNativeRMF(pditem, cbBuffer, pBuffer);
```

DrgQueryNativeRMF Parameter - pditem

pditem ([PDRAGITEM](#)) - input
Pointer to the [DRAGITEM](#) structure.

Pointer to the [DRAGITEM](#) structure whose native rendering mechanism and format are to be obtained.

DrgQueryNativeRMF Parameter - cbBuffer

cbBuffer ([ULONG](#)) - input
Maximum number of bytes to copy to the buffer.

DrgQueryNativeRMF Parameter - pBuffer

pBuffer ([PCHAR](#)) - output
Pointer to the buffer in which the null-terminated string is to be returned.

DrgQueryNativeRMF Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgQueryNativeRMF - Parameters

pditem ([PDRAGITEM](#)) - input
Pointer to the [DRAGITEM](#) structure.

Pointer to the [DRAGITEM](#) structure whose native rendering mechanism and format are to be obtained.

cbBuffer ([ULONG](#)) - input
Maximum number of bytes to copy to the buffer.

pBuffer ([PCHAR](#)) - output
Pointer to the buffer in which the null-terminated string is to be returned.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgQueryNativeRMF - Remarks

If the rendering mechanism and format string for the object are NULL, FALSE is returned. If TRUE is returned, the format of the string is:

```
(mechanism(1)[,mechanism(n)...]),  
(format(1)[,format(n)...])
```

The native rendering mechanism and format are the first ordered pair, or the first ordered pair produced by a cross product, in the string associated with *hstrRMF* in the [DRAGITEM](#) structure.

[DrgQueryNativeRMFLen](#) can be used to determine the size of the buffer required to hold the string returned by this function.

DrgQueryNativeRMF - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_PARAMETERS (0x1208)

An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32 768 to +32 767 cannot be converted to a [SHORT](#), and a negative number cannot be converted to a [ULONG](#) or [USHORT](#).

DrgQueryNativeRMF - Related Functions

Prerequisite Functions

- [DrgQueryNativeRMFLen](#)

Related Functions

- [DrgVerifyNativeRMF](#)
-

DrgQueryNativeRMF - Example Code

This example shows how to obtain the window handle of the source of a drag item.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#define INCL_DOSMEMMGR  /* Memory Management Functions for      */
                        /* DosSubAlloc                          */
#include <OS2.H>

DRAGITEM ditem;
PVOID    pMem;
PSZ      pszBuffer;
ULONG    cb;
BOOL     rc, fResult;

cb = DrgQueryNativeRMFLen(&ditem) + 1;
rc = DosSubAlloc(pMem, (PVOID *) pszBuffer, cb);

if (!rc)
{
    fResult = DrgQueryNativeRMF(&ditem, cb, pszBuffer);
}
```

DrgQueryNativeRMF - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgQueryNativeRMFLen

DrgQueryNativeRMFLen - Syntax

Obtains the length of the string representing the native rendering mechanism and format of the dragged object.

```
#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGITEM    pditem;    /* Pointer to the DRAGITEM structure whose native rendering mechanism and format string
ULONG        ulLength;  /* String length of the ordered pair. */

ulLength = DrgQueryNativeRMFLen(pditem);
```

DrgQueryNativeRMFLen Parameter - pditem

pditem ([PDRAGITEM](#)) - input
Pointer to the [DRAGITEM](#) structure whose native rendering mechanism and format string length are to be obtained.

DrgQueryNativeRMFLen Return Value - ulLength

ulLength ([ULONG](#)) - returns
String length of the ordered pair.

0	Error occurred.
Other	String length of the ordered pair, excluding the null-terminating byte.

DrgQueryNativeRMFLen - Parameters

pditem ([PDRAGITEM](#)) - input
Pointer to the [DRAGITEM](#) structure whose native rendering mechanism and format string length are to be obtained.

uiLength ([ULONG](#)) - returns
String length of the ordered pair.

- 0
Error occurred.
- Other
String length of the ordered pair, excluding the null-terminating byte.

DrgQueryNativeRMFLen - Remarks

This function is used to determine the size of the buffer that contains the string representing the native rendering mechanism and format of the dragged object.

If the input string handle is NULLHANDLE or not valid, a length of 0 is returned.

DrgQueryNativeRMFLen - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_PARAMETERS (0x1208)
An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32 768 to +32 767 cannot be converted to a [SHORT](#), and a negative number cannot be converted to a [ULONG](#) or [USHORT](#).

DrgQueryNativeRMFLen - Related Functions

- Related Functions**
- [DrgQueryNativeRMF](#)

DrgQueryNativeRMFLen - Example Code

This example shows how to obtain the window handle of the source of a drag item.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#define INCL_DOSMEMMGR /* Memory Management Functions for */
                        /* DosSubAlloc */
#include <OS2.H>

DRAGITEM ditem;
PVOID    pMem;
PSZ      pszBuffer;
ULONG    cb;
BOOL     rc, fResult;
```

```

cb = DrgQueryNativeRMFLen(&ditem) + 1;

rc = DosSubAlloc(pMem, (PVOID *) pszBuffer, cb);

if (!rc)
{
    fResult = DrgQueryNativeRMF(&ditem, cb, pszBuffer);
}

```

DrgQueryNativeRMFLen - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgQueryStrName

DrgQueryStrName - Syntax

This function gets the contents of a string associated with a string handle.

```

#define INCL_WINSTDDRAG
#include <os2.h>

HSTR      hstr;      /* The handle must have been created with DrgAddStrHandle. */
ULONG     cbBuffer;  /* Maximum number of bytes to copy into pBuffer. */
PSZ       pBuffer;   /* Buffer where the null-terminated string is returned. */
ULONG     ulLength;  /* Number of bytes written to pBuffer. */

ulLength = DrgQueryStrName(hstr, cbBuffer,
                           pBuffer);

```

DrgQueryStrName Parameter - hstr

hstr ([HSTR](#)) - input

The handle must have been created with [DrgAddStrHandle](#).

DrgQueryStrName Parameter - cbBuffer

cbBuffer ([ULONG](#)) - input

Maximum number of bytes to copy into *pBuffer*.

Must be greater than 0. Otherwise, an error is returned.

DrgQueryStrName Parameter - pBuffer

pBuffer ([PSZ](#)) - output

Buffer where the null-terminated string is returned.

DrgQueryStrName Return Value - ulLength

ulLength ([ULONG](#)) - returns

Number of bytes written to *pBuffer*.

DrgQueryStrName - Parameters

hstr ([HSTR](#)) - input

The handle must have been created with [DrgAddStrHandle](#).

cbBuffer ([ULONG](#)) - input

Maximum number of bytes to copy into *pBuffer*.

Must be greater than 0. Otherwise, an error is returned.

pBuffer ([PSZ](#)) - output

Buffer where the null-terminated string is returned.

ulLength ([ULONG](#)) - returns

Number of bytes written to *pBuffer*.

DrgQueryStrName - Remarks

This function should be called whenever the contents of a string referenced by a drag string handle are required. If the input string handle is

NULLHANDLE or not valid, a null string is returned.

DrgQueryStrName - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_PARAMETERS (0x1208)

An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32 768 to +32 767 cannot be converted to a [SHORT](#), and a negative number cannot be converted to a [ULONG](#) or [USHORT](#).

DrgQueryStrName - Related Functions

Prerequisite Functions

- [DrgQueryStrNameLen](#)

Related Functions

- [DrgAddStrHandle](#)
-

DrgQueryStrName - Example Code

This example shows how to obtain the contents of a string given that the string handle is known. The string handle must have been originally created with the [DrgAddStrHandle](#) function.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#define INCL_DOSMEMMGR /* Memory Management Functions for */
                        /* DosAllocMem */
#include <OS2.H>

HSTR    hstr;          /* Handle to a string. The handle must */
                        /* have been created with */
                        /* DrgAddStrHandle. */
PSZ     pBuffer;       /* Buffer where the null-terminated */
                        /* string is returned */
ULONG   ulStrlen;      /* String length */
ULONG   ulBytesRead;   /* Number of bytes read */
ULONG   rc;            /* Return code */

ulStrlen = DrgQueryStrNameLen(hstr) + 1;

rc = DosAllocMem((PVOID *) pBuffer,
                (LONG)ulStrlen,
                fPERM |
                PAG_COMMIT);

/*****
/* The ulBytesRead parameter contains the number of bytes
/* actually written to the memory pointed to by pBuffer
*****/
ulBytesRead = DrgQueryStrName(hstr,
                            ulStrlen, /* Number of bytes to copy */
                            pBuffer);
```

DrgQueryStrName - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgQueryStrNameLen

DrgQueryStrNameLen - Syntax

This function gets the length of a string associated with a string handle.

```
#define INCL_WINSTDDRAG
#include <os2.h>

HSTR      hstr;      /* String handle. */
ULONG     cLength;   /* Length of the string associated with hstr. */

cLength = DrgQueryStrNameLen(hstr);
```

DrgQueryStrNameLen Parameter - hstr

hstr ([HSTR](#)) - input
String handle.

The handle must be created with [DrgAddStrHandle](#).

DrgQueryStrNameLen Return Value - cLength

cLength ([ULONG](#)) - returns
Length of the string associated with *hstr*.

0	The string handle is NULLHANDLE or is not valid.
Other	The length of the string associated with the string handle, excluding the null terminating byte.

DrgQueryStrNameLen - Parameters

hstr ([HSTR](#)) - input
String handle.

The handle must be created with [DrgAddStrHandle](#).

cLength ([ULONG](#)) - returns
Length of the string associated with *hstr*.

0	The string handle is NULLHANDLE or is not valid.
Other	The length of the string associated with the string handle, excluding the null terminating byte.

DrgQueryStrNameLen - Remarks

This function should be called before calling the [DrgQueryStrName](#) function. It is used to determine and allocate the buffer size for the string associated with the string handle. If the input string handle is NULLHANDLE or not valid, a length of 0 is returned.

DrgQueryStrNameLen - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_PARAMETERS (0x1208)
An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32 768 to +32 767 cannot be converted to a [SHORT](#), and a negative number cannot be converted to a [ULONG](#) or [USHORT](#).

DrgQueryStrNameLen - Related Functions

Related Functions

- [DrgQueryStrName](#)

DrgQueryStrNameLen - Example Code

This example shows how to obtain the length of a string given that the string handle is known. The string handle must have been originally

created with the [DrgAddStrHandle](#) function.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#define INCL_DOSMEMMGR /* Memory Management Functions for */
                        /* DosAllocMem */
#include <OS2.H>

HSTR    hstr;          /* Handle to a string. The handle must */
                        /* have been created with */
                        /* DrgAddStrHandle. */
PSZ      pBuffer;      /* Buffer where the null-terminated */
                        /* string is returned */
ULONG    ulStrlen;      /* String length */
ULONG    ulBytesRead;   /* Number of bytes read */
ULONG    rc;           /* Return code */

ulStrlen = DrgQueryStrNameLen(hstr) + 1;

rc = DosAllocMem((PVOID *) pBuffer,
                (LONG)ulStrlen,
                fPERM |
                PAG_COMMIT);

/*****
/* The ulBytesRead parameter contains the number of bytes */
/* actually written to the memory pointed to by pBuffer */
*****/
ulBytesRead = DrgQueryStrName(hstr,
                            ulStrlen, /* Number of bytes to copy */
                            pBuffer);
```

DrgQueryStrNameLen - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgQueryTrueType

DrgQueryTrueType - Syntax

This function obtains the true type of a dragged object.

```
#define INCL_WINSTDDRAG
#include <os2.h>

PDRAITEM    pditem; /* Pointer to the DRAGITEM structure whose type is to be obtained. */
```

```
ULONG      cbBuffer; /* Maximum number of bytes to copy to pBuffer. Must be > 0. */
PSZ        pBuffer; /* Buffer in which the null-terminated string is to be returned. */
BOOL       rc;      /* Success indicator. */

rc = DrgQueryTrueType(pditem, cbBuffer, pBuffer);
```

DrgQueryTrueType Parameter - pditem

pditem (**PDRAGITEM**) - input
Pointer to the **DRAGITEM** structure whose type is to be obtained.

DrgQueryTrueType Parameter - cbBuffer

cbBuffer (**ULONG**) - input
Maximum number of bytes to copy to *pBuffer*. Must be > 0.

DrgQueryTrueType Parameter - pBuffer

pBuffer (**PSZ**) - output
Buffer in which the null-terminated string is to be returned.

DrgQueryTrueType Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgQueryTrueType - Parameters

pditem (**PDRAGITEM**) - input
Pointer to the **DRAGITEM** structure whose type is to be obtained.

cbBuffer ([ULONG](#)) - input
Maximum number of bytes to copy to *pBuffer*. Must be > 0.

pBuffer ([PSZ](#)) - output
Buffer in which the null-terminated string is to be returned.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	
FALSE	Successful completion.
	Error occurred.

DrgQueryTrueType - Remarks

The true type of an object is the first type in the string referenced by *hstrType* in the [DRAGITEM](#) structure.

This function can be called after calling the [DrgQueryTrueTypeLen](#) function. If the type string for the object is NULLHANDLE, FALSE is returned.

DrgQueryTrueType - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_PARAMETERS (0x1208)
An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32 768 to +32 767 cannot be converted to a [SHORT](#), and a negative number cannot be converted to a [ULONG](#) or [USHORT](#).

DrgQueryTrueType - Related Functions

Prerequisite Functions

- [DrgQueryTrueTypeLen](#)

Related Functions

- [DrgVerifyTrueType](#)

DrgQueryTrueType - Example Code

This example shows how to obtain the true type of an object.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <OS2.H>

BOOL fSuccess;          /* Return value */
DRAGITEM Dragitem;      /* DRAGITEM structure whose true type */
```

```

/* is to be obtained */
char szBuffer[32]; /* Buffer in which the null-terminated */
/* string is to be returned */

fSuccess = DrgQueryTrueType(&Dragitem,
                             sizeof(szBuffer),
                             szBuffer);

```

DrgQueryTrueType - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgQueryTrueTypeLen

DrgQueryTrueTypeLen - Syntax

This function obtains the length of the string that represents the true type of a dragged object.

```

#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGITEM    pditem; /* Pointer to the DRAGITEM structure whose type length is to be obtained. */
ULONG        ulLength; /* String length of the first element of the character string associated with hstrType

ulLength = DrgQueryTrueTypeLen(pditem);

```

DrgQueryTrueTypeLen Parameter - pditem

pditem ([PDRAGITEM](#)) - input
 Pointer to the [DRAGITEM](#) structure whose type length is to be obtained.

DrgQueryTrueTypeLen Return Value - ulLength

ulLength ([ULONG](#)) - returns
String length of the first element of the character string associated with *hstrType*.

0	Error occurred.
Other	The length of the first element of the character string associated with <i>hstrType</i> , excluding the null-terminating byte.

DrgQueryTrueTypeLen - Parameters

pditem ([PDRAGITEM](#)) - input
Pointer to the [DRAGITEM](#) structure whose type length is to be obtained.

ulLength ([ULONG](#)) - returns
String length of the first element of the character string associated with *hstrType*.

0	Error occurred.
Other	The length of the first element of the character string associated with <i>hstrType</i> , excluding the null-terminating byte.

DrgQueryTrueTypeLen - Remarks

This function can be used to determine the buffer size to allocate for the string representing the true type of a dragged object. The true type of an object is the first type in the type string referenced by *hstrType* in the [DRAGITEM](#) structure.

This function can be called before calling the [DrgQueryTrueType](#) function.

If the input string handle is NULLHANDLE or not valid, a length of 0 is returned.

DrgQueryTrueTypeLen - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_PARAMETERS (0x1208)
An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32 768 to +32 767 cannot be converted to a [SHORT](#), and a negative number cannot be converted to a [ULONG](#) or [USHORT](#).

DrgQueryTrueTypeLen - Related Functions

- Related Functions**
- [DrgQueryTrueType](#)

DrgQueryTrueTypeLen - Example Code

This example shows how to obtain the length of the true type string with the DrgQueryTrueTypeLen function.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#define INCL_DOSMEMMGR /* Memory Management Functions for */
                        /* DosAllocMem */
#include <OS2.H>

PSZ      pszBuffer; /* Buffer in which the DRAGITEM */
                        /* structure is stored */
BOOL     fSuccess; /* Return value */
DRAGITEM Dragitem; /* DRAGITEM structure whose true type */
                        /* length is to be obtained */
ULONG    rc; /* Return code */
ULONG    ulLength; /* String length of dragged object */

ulLength = DrgQueryTrueTypeLen(&Dragitem) + 1;

rc = DosAllocMem((PVOID *) pszBuffer, ulLength, fPERM);

fSuccess = DrgQueryTrueType(&Dragitem, ulLength, pszBuffer);
```

DrgQueryTrueTypeLen - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgReallocDragInfo

DrgReallocDragInfo - Syntax

This function releases the current [DRAGINFO](#) structure and reallocates a new one.

```
#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGINFO pdrinfoOld; /* Pointer to the current DRAGINFO structure. */
ULONG     cditem; /* Number of DRAGITEM structures to be allocated. */
PDRAGINFO pdrinfoCurrent; /* Pointer to a newly allocated DRAGINFO structure. */
```

```
pdinfoCurrent = DrgReallocDragInfo(pdinfoOld,  
                                   cditem);
```

DrgReallocDragInfo Parameter - pdinfoOld

pdinfoOld ([PDRAGINFO](#)) - input
Pointer to the current [DRAGINFO](#) structure.

DrgReallocDragInfo Parameter - cditem

cditem ([ULONG](#)) - input
Number of [DRAGITEM](#) structures to be allocated.

DrgReallocDragInfo Return Value - pdinfoCurrent

pdinfoCurrent ([PDRAGINFO](#)) - returns
Pointer to a newly allocated [DRAGINFO](#) structure.

DrgReallocDragInfo - Parameters

pdinfoOld ([PDRAGINFO](#)) - input
Pointer to the current [DRAGINFO](#) structure.

cditem ([ULONG](#)) - input
Number of [DRAGITEM](#) structures to be allocated.

pdinfoCurrent ([PDRAGINFO](#)) - returns
Pointer to a newly allocated [DRAGINFO](#) structure.

DrgReallocDragInfo - Remarks

It is necessary to call `DrgReallocDragInfo` anytime objects are added or deleted from the current lazy drag set. This function unconditionally frees the old [DRAGINFO](#) structure, reallocates a new [DRAGINFO](#) structure, and returns a pointer to the new structure.

Note: This function does not check if the source and target window handles are different; it unconditionally frees the [DRAGINFO](#) structure passed to it.

DrgReallocDragInfo - Related Functions

Related Functions

- [DrgAllocDraginfo](#)
-

DrgReallocDragInfo - Example Code

This example uses DrgReallocDragInfo to reallocate the [DRAGINFO](#) structure when an object is picked up or added to the lazy drag set. It checks whether a lazy drag operation is already in progress, and if so, adds one to the number of objects currently being dragged and calls DrgReallocDragInfo to obtain a new [DRAGINFO](#) structure with the required number of [DRAGITEM](#) structures.

```
#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGINFO pdinfoCurrent; /* Pointer to the current DRAGINFO */
PDRAGINFO pdinfoOld; /* Pointer to the DRAGINFO to be freed */
ULONG cditem; /* Number of DRAGITEMS */
.
.
.
case WM_PICKUP:

    if (DrgQueryDragStatus() & DGS_LAZYDRAGINPROGRESS)
    {
        /* Get a pinter to the current DRAGINFO structure */
        pdinfoOld=DrgQueryDraginfoPtr(NULL);

        /* Add space for one more DRAGITEM */
        cditem=pdinfoOld->cditem+1;

        /* Reallocate the DRAGINFO */
        pdinfoCurrent=DrgReallocDraginfo(pdinfoOld, cditem);
        if(pdinfoCurrent)
        {
            /* Continue the lazy drag operation */
            DrgLazyDrag( ... )
        }
    }
}
```

DrgReallocDragInfo - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgReleasePS

DrgReleasePS - Syntax

This function releases a presentation space obtained by using the [DrgGetPS](#) function.

```
#define INCL_WINSTDDRAG
#include <os2.h>

HPS    hps; /* Handle of the presentation space to release. */
BOOL    rc; /* Success indicator. */

rc = DrgReleasePS(hps);
```

DrgReleasePS Parameter - hps

hps ([HPS](#)) - input
Handle of the presentation space to release.

DrgReleasePS Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgReleasePS - Parameters

hps ([HPS](#)) - input
Handle of the presentation space to release.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
------	------------------------

FALSE

Error occurred.

DrgReleasePS - Remarks

Only presentation spaces created with [DrgGetPS](#) can be released using this function.

The presentation-space handle should not be used after this function.

DrgReleasePS - Errors

Possible returns from [WinGetLastError](#)

PMERR_INV_HPS (0x207F)

An invalid presentation-space handle was specified.

PMERR_NOT_DRAGGING (0x1f00)

A drag operation is not in progress at this time.

DrgReleasePS - Related Functions

Prerequisite Functions

- [DrgGetPS](#)

DrgReleasePS - Example Code

In this example the presentation space handle is retrieved, a bit map is loaded, and the presentation space is released with the `DrgReleasePS` function.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>
#define ID_BITMAP 255
HPS hps;
HWND hwnd;

case DM_DRAGOVER:
    hps = DrgGetPS(hwnd);

    DrawTargetEmphasis(hps, hwnd);
    DrgReleasePS(hps);
```

DrgReleasePS - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgSendTransferMsg

DrgSendTransferMsg - Syntax

This function sends a message to the other application involved in the direct manipulation operation.

```
#define INCL_WINSTDDRAG
#include <os2.h>

HWND      hwndTo;      /* Window handle to which the message is to be sent. */
ULONG     ulMsgid;     /* Identifier of the message to be sent. */
MPARAM    mpParam1;    /* First message parameter. */
MPARAM    mpParam2;    /* Second message parameter. */
MRESULT   mresReply;   /* Message-return data. */

mresReply = DrgSendTransferMsg(hwndTo, ulMsgid,
                               mpParam1, mpParam2);
```

DrgSendTransferMsg Parameter - hwndTo

hwndTo ([HWND](#)) - input

Window handle to which the message is to be sent.

Target

hwndItem in the [DRAGITEM](#) structure.

Source

hwndClient in the [DRAGTRANSFER](#) structure.

DrgSendTransferMsg Parameter - ulMsgid

ulMsgid ([ULONG](#)) - input

Identifier of the message to be sent.

Valid messages are:

DM_ENDCONVERSATION
DM_RENDER
DM_RENDERPREPARE

DrgSendTransferMsg Parameter - mpParam1

mpParam1 ([MPARAM](#)) - input
First message parameter.

DrgSendTransferMsg Parameter - mpParam2

mpParam2 ([MPARAM](#)) - input
Second message parameter.

DrgSendTransferMsg Return Value - mresReply

mresReply ([MRESULT](#)) - returns
Message-return data.

DrgSendTransferMsg - Parameters

hwndTo ([HWND](#)) - input
Window handle to which the message is to be sent.

Target
hwndItem in the [DRAGITEM](#) structure.

Source
hwndClient in the [DRAGTRANSFER](#) structure.

ulMsgid ([ULONG](#)) - input
Identifier of the message to be sent.

Valid messages are:

DM_ENDCONVERSATION
DM_RENDER
DM_RENDERPREPARE

mpParam1 ([MPARAM](#)) - input
First message parameter.

mpParam2 ([MPARAM](#)) - input
Second message parameter.

mresReply ([MRESULT](#)) - returns
Message-return data.

DrgSendTransferMsg - Remarks

If the message to be sent is [DM_RENDER](#) or [DM_RENDERCOMPLETE](#), the *fsReply* field in [DRAGTRANSFER](#) is set to 0 before the message is sent. If the message cannot be sent, FALSE is returned.

When the message to be sent is [DM_RENDER](#), DosGiveSeg is called. DosGiveSeg gives access to the [DRAGTRANSFER](#) structure to the process that owns the window indicated by *hwndTo*. The use count for the segment in which the [DRAGTRANSFER](#) structure exists is incremented.

The process to which the message is being sent must call [DrgFreeDragtransfer](#) for the [DRAGTRANSFER](#) structure before the segment can be freed.

DrgSendTransferMsg - Related Functions

Related Functions

- [DrgPostTransferMsg](#)

DrgSendTransferMsg - Example Code

This function is used to send a message from one window to another when a direct manipulation is in progress. In this example, the function is used to inform the target that the operation is complete and successful.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

PDRAGINFO pdinfo;
MPARAM    mp1;
TID       tid;

case DM_DROP:
    pdinfo = (PDRAGINFO) mp1;

    /******
    /* If this is a copy operation, spawn a thread to do the copy */
    /******
    if (pdinfo->usOperation == DO_COPY)
    {
        DosCreateThread (&tid, CopyThread, pdinfo, FALSE, 4096);
    }
    break;

void Copy Thread (PDRAGINFO pdinfo)
{
    PDRAGITEM pditem;
    USHORT    i;
    ULONG     flResult;
    HAB       hab;
    HMQ       hmq;
    char      szSource[CCH_MAXPATH];
    char      szTarget[CCH_MAXPATH];
```

```

/*****
/* DrgSendTransferMsg needs a message queue, so create one for */
/* this thread */
/*****
hab = WinInitialize (0);
hmq = WinCreateMsgQueue (hab, 0);

/*****
/* Try to copy each item that was dragged */
/*****
for (i = 0; i < pdinfo->cditem; i++)
{
    /*****
    /* Get a pointer to the DRAGITEM */
    /*****
    pditem = DrgQueryDragitemPtr (pdinfo, i);

    /*****
    /* If we could query the source and target names, and the */
    /* copy was successful, return success */
    /*****
    if (DrgQueryStrName (pditem->hstrSourceName, sizeof (szSource),
        szSource)
        DrgQueryStrName (pditem->hstrTargetName, sizeof (szTarget),
        szTarget)
        !DosCopy (szSource, szTarget, 0))
    {
        flResult = DMFL_TARGETSUCCESSFUL;
    }

    /*****
    /* Otherwise, return failure */
    /*****
    else
    {
        flResult = DMFL_TARGETFAIL;
    }

    /*****
    /* Let the source know we're done with this item */
    /*****
    DrgSendTransferMsg (pditem->hwndItem, DM_ENDCONVERSATION,
        (MPARAM) pditem->ulItemID,
        (MPARAM) flResult);
}

WinDestroyMsgQueue (hmq);
WinTerminate (hab);
}

```

DrgSendTransferMsg - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

DrgSetDragImage

DrgSetDragImage - Syntax

This function sets the image that is being dragged.

```
#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGINFO    pdinfo; /* Pointer to the DRAGINFO structure representing the drag operation for which the pointer is to be set. */
PDRAGIMAGE    pdimg; /* Pointer to an array of DRAGIMAGE structures. */
ULONG        cdimg; /* Number of DRAGIMAGE structures in the pdimg array. */
PVOID        pRsvd; /* Reserved value, must be NULL. */
BOOL         rc; /* Success indicator. */

rc = DrgSetDragImage(pdinfo, pdimg, cdimg,
                    pRsvd);
```

DrgSetDragImage Parameter - pdinfo

pdinfo ([PDRAGINFO](#)) - input
Pointer to the [DRAGINFO](#) structure representing the drag operation for which the pointer is to be set.

DrgSetDragImage Parameter - pdimg

pdimg ([PDRAGIMAGE](#)) - input
Pointer to an array of [DRAGIMAGE](#) structures.

These structures describe the images to be drawn under the pointer during the drag.

DrgSetDragImage Parameter - cdimg

cdimg ([ULONG](#)) - input
Number of [DRAGIMAGE](#) structures in the *pdimg* array.

DrgSetDragImage Parameter - pRsvd

pRsvd ([PVOID](#)) - input
Reserved value, must be NULL.

DrgSetDragImage Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgSetDragImage - Parameters

pdinfo ([PDRAGINFO](#)) - input
Pointer to the [DRAGINFO](#) structure representing the drag operation for which the pointer is to be set.

pdimg ([PDRAGIMAGE](#)) - input
Pointer to an array of [DRAGIMAGE](#) structures.

These structures describe the images to be drawn under the pointer during the drag.

cdimg ([ULONG](#)) - input
Number of [DRAGIMAGE](#) structures in the *pdimg* array.

pRsvd ([PVOID](#)) - input
Reserved value, must be NULL.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgSetDragImage - Remarks

The image that is set with DrgSetDragImage is used only while the pointer is over the target that made the call. If the pointer leaves the original target, the new target can specify an image by calling DrgSetDragImage.

If the new target does not call DrgSetDragImage, the original image that was supplied on the call to [DrgDrag](#) is used.

DrgSetDragImage - Errors

Possible returns from [WinGetLastError](#)

PMERR_ACCESS_DENIED (0x150D)

The memory block was not allocated properly.

PMERR_INVALID_PARAMETERS (0x1208)

An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32 768 to +32 767 cannot be converted to a [SHORT](#), and a negative number cannot be converted to a [ULONG](#) or [USHORT](#).

PMERR_INSUFFICIENT_MEMORY (0x203E)

The operation terminated through insufficient memory.

DrgSetDragImage - Related Functions

Related Functions

- [DrgSetDragPointer](#)
-

DrgSetDragImage - Example Code

This example sets the icon image that is displayed during a direct manipulation operation.

```
#define INCL_GPIBITMAPS /* GPI Bit Map Functions */
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>
#define ID_BITMAP 257 /* .rc file: "bitmap 257 drgimage.bmp" */
HPS hps; /* Presentation space handle */
BOOL flResult;
HAB hab;
PDRAGINFO pdinfo;
DRAGIMAGE dimg;
HBITMAP hbm; /* Bit-map handle */
HWND hwnd;

/*****
 * Load a bit map for use as a drag image
 *****/

case WM_CREATE:
    hps = WinGetPS(hwnd);

    hbm = GpiLoadBitmap(hps, 0L, ID_BITMAP, 20L, 20L);
    WinReleasePS(hps);
    break;

case DM_DRAGOVER:
/*****
 * Initialize the DRAGIMAGE structure
 *****/

    dimg.cb = sizeof(DRAGIMAGE); /* Size control block */
    dimg.cptl = 0;
    dimg.hImage = hbm; /* Image handle passed to */
    /* DrgDrag */
    dimg.sizlStretch.cx = 20L; /* Size to stretch ico or */
    dimg.sizlStretch.cy = 20L; /* bmp to */
    dimg.fl = DRG_BITMAP |
        DRG_STRETCH; /* Stretch to size specified */
    dimg.cxOffset = 0; /* Offset of the origin of */
    dimg.cyOffset = 0; /* the image from the pointer*/
    /* hotspot */

/*****
```

```
/* Set the drag image */
/*****
flResult= DrgSetDragImage(pdinfo,&dimg,(ULONG)sizeof(dimg), NULL);
```

DrgSetDragImage - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

DrgSetDragitem

DrgSetDragitem - Syntax

This function sets the values in a [DRAGITEM](#) structure.

```
#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGINFO    pdinfo;    /* Pointer to the DRAGINFO structure in which to place the DRAGITEM. */
PDRAGITEM    pditem;    /* Pointer to the DRAGITEM structure to place in DRAGINFO. */
ULONG        cbBuffer;  /* Size of the DRAGITEM addressed by pditem. */
ULONG        iItem;     /* Zero-based index of the DRAGITEM to be set. */
BOOL         rc;        /* Success indicator. */

rc = DrgSetDragitem(pdinfo, pditem, cbBuffer,
                    iItem);
```

DrgSetDragitem Parameter - pdinfo

pdinfo ([PDRAGINFO](#)) - input

Pointer to the [DRAGINFO](#) structure in which to place the [DRAGITEM](#).

DrgSetDragitem Parameter - pditem

pditem (**PDRAGITEM**) - input
Pointer to the **DRAGITEM** structure to place in **DRAGINFO**.

DrgSetDragitem Parameter - cbBuffer

cbBuffer (**ULONG**) - input
Size of the **DRAGITEM** addressed by *pditem*.

DrgSetDragitem Parameter - iltem

iltem (**ULONG**) - input
Zero-based index of the **DRAGITEM** to be set.

DrgSetDragitem Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgSetDragitem - Parameters

pdinfo (**PDRAGINFO**) - input
Pointer to the **DRAGINFO** structure in which to place the **DRAGITEM**.

pditem (**PDRAGITEM**) - input
Pointer to the **DRAGITEM** structure to place in **DRAGINFO**.

cbBuffer (**ULONG**) - input
Size of the **DRAGITEM** addressed by *pditem*.

iltem (**ULONG**) - input
Zero-based index of the **DRAGITEM** to be set.

rc (**BOOL**) - returns
Success indicator.

TRUE

FALSE Successful completion.
 Error occurred.

DrgSetDragitem - Remarks

This function is used to initialize the [DRAGINFO](#) structure before calling [DrgDrag](#).

This function is used only by the source of the drag, not by the target.

DrgSetDragitem - Related Functions

Related Functions

- [DrgQueryDragitem](#)
-

DrgSetDragitem - Example Code

This example shows a direct manipulation operation between two windows. The actual operation, copying the CONFIG.SYS file to C:\OS2\CONFIG.SYS, is visually represented by a drag and drop of an icon.

```
#define INCL_GPIBITMAPS /* GPI Bit Map Functions */
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#define INCL_DOSFILEMGR /* File Management Functions */
#define INCL_WININPUT /* Window Input Functions */
#include <os2.h>
#include <string.h>
#define ID_WINDOW 255
#define ID_ITEM 256
#define ID_BITMAP 257 /* .rc file: "bitmap 257 drgimage.bmp" */
HPS hps; /* Presentation space handle */
BOOL flResult;
HAB hab;
PDRAGINFO pdinfo;
DRAGITEM ditem;
DRAGIMAGE dimg;
PDRAGITEM pditem;
HBITMAP hbm; /* Bit-map handle */
HPOINTER hptr; /* Pointer bit-map handle */
HWND hwndDrop;
HWND hwnd;
MPARAM mp1;
char szBuffer[32]; /* Buffer where intersection string */
/* is returned */
char szSource[32];
char szTarget[32];

/*****
/* Inside ClientWindowProc of Source Window
*****/

case WM_BEGINDRAG:

/*****
/* Initialize the DRAGITEM structure
*****/

ditem.hwndItem = hwnd; /* Conversation partner */
```



```

        ditem.ulItemID = ID_ITEM; /* Identifies item being dragged */
        ditem.hstrType = DrgAddStrHandle(DRT_TEXT); /* Text item */
        ditem.hstrRMF = DrgAddStrHandle("<DRM_OS2FILE,DRF_TEXT>");
        ditem.hstrContainerName = DrgAddStrHandle("C:\\");
        ditem.hstrSourceName = DrgAddStrHandle("C:\\CONFIG.SYS");
        ditem.hstrTargetName = DrgAddStrHandle("C:\\OS2\\CONFIG.SYS");
        ditem.cxOffset = 0; ditem.cyOffset = 0;
        ditem.fsControl = 0; ditem.fsSupportedOps = 0;

/*****
/* Create the DRAGINFO structure */
*****/

        pdinfo = DrgAllocDraginfo(1);

/*****
/* Initialize the DRAGIMAGE structure */
*****/

        dimg.cb = sizeof(DRAGIMAGE); /* Size control block */
        dimg.cptl = 0;
        dimg.hImage = hbm; /* Image handle passed to */
        /* DrgDrag */
        dimg.sizlStretch.cx = 20L; /* Size to stretch ico or */
        dimg.sizlStretch.cy = 20L; /* bmp to */
        dimg.fl = DRG_BITMAP |
                DRG_STRETCH; /* Stretch to size specified */
        dimg.cxOffset = 0; /* Offset of the origin of the */
        dimg.cyOffset = 0; /* image from the pointer */
        /* hotspot */

flResult= DrgSetDragitem(pdinfo, &ditem, (ULONG)sizeof(ditem), 0);

/*****
/* Perform the drag operation: */
*****/

        hwndDrop = DrgDrag(hwnd, /* Source of the drag */
                pdinfo, /* Pointer to DRAGINFO structure */
                (PDRAGIMAGE)&dimg, /* Drag image */
                1, /* Size of the pdimg array */
                VK_ENGDRAW, /* Release of drag button */
                /* terminates the drag */
                NULL); /* Reserved */

/*****
/* Inside ClientWindowProc of Target Window */
*****/

        case DM_DRAGOVER:

                pdinfo = MPFROMP(mpl);
                pditem = DrgQueryDragitemPtr(pdinfo,0);

                flResult = DrgVerifyTrueType(pditem,"DRF_TEXT");

                if(!flResult)

/*****
/* Inform the application that you will accept the drop */
*****/

                return(MRFROM2SHORT(DOR_DROP,DO_COPY));

        case DM_DROP:

                pdinfo = MPFROMP(mpl);
                pditem = DrgQueryDragitemPtr(pdinfo,0);

/*****
/* Perform the operation represented by the direct manipulation */
*****/

        DrgQueryStrName(pditem->hstrSourceName,sizeof(szSource),szSource);
        DrgQueryStrName(pditem->hstrTargetName,sizeof(szTarget),szTarget);
        flResult = DosCopy(szSource,szTarget,0L);

/*****
/* If operation is successful, return DMFL_TARGETSUCCESSFUL */
*****/

```

```

if(!flResult)
{
    DrgSendTransferMsg(pditem->hwndItem,
                      DM_ENDCONVERSATION,
                      MPFROMLONG(pditem->ulItemID),
                      MPFROMLONG(DMFL_TARGETSUCCESSFUL));
}

/*****
/* Otherwise, return DMFL_TARGETFAIL */
*****/

else
{
    DrgSendTransferMsg(pditem->hwndItem,
                      DM_ENDCONVERSATION,
                      MPFROMLONG(pditem->ulItemID),
                      MPFROMLONG(DMFL_TARGETFAIL));
}

```

DrgSetDragitem - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgSetDragPointer

DrgSetDragPointer - Syntax

This function sets the pointer to be used while over the current target.

```

#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGINFO    pdinfo; /* Pointer to the DRAGINFO structure to be used for this drag. */
HPOINTER     hptr;   /* Handle to the pointer to use. */
BOOL         rc;      /* Success indicator. */

rc = DrgSetDragPointer(pdinfo, hptr);

```

DrgSetDragPointer Parameter - pdinfo

pdinfo (**PDRAGINFO**) - input
Pointer to the **DRAGINFO** structure to be used for this drag.

DrgSetDragPointer Parameter - hptr

hptr (**HPOINTER**) - input
Handle to the pointer to use.

DrgSetDragPointer Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgSetDragPointer - Parameters

pdinfo (**PDRAGINFO**) - input
Pointer to the **DRAGINFO** structure to be used for this drag.

hptr (**HPOINTER**) - input
Handle to the pointer to use.

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgSetDragPointer - Remarks

This function sets the pointer to be used to indicate the hot spot while dragging over the current target.

The pointer that is set with DrgSetDragPointer is used only while it is over the current target. The pointer is reset to the default when a new target is dragged over.

This function can be used by an application to provide meaningful augmentation emphasis for an operation that is unknown to the system

(for example, swap).

When the drag pointer is successfully set, TRUE is returned.

DrgSetDragPointer - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HPTR (0x101B)
An invalid pointer handle was specified.

DrgSetDragPointer - Related Functions

Related Functions

- [DrgSetDragImage](#)

DrgSetDragPointer - Example Code

This example uses the DrgSetDragPointer function to set the image used for the pointer while the pointer is over the target during a direct manipulation operation.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <OS2.H>
BOOL      flResult;
PDRAGITEM pditem;
HPOINTER  hptrCrossHair;
MPARAM    mp1;
char      szBuffer[32];

case DM_DRAGOVER:
    DrgSetDragPointer ((PDRAGINFO) mp1, hptrCrossHair);
```

DrgSetDragPointer - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgVerifyNativeRMF

DrgVerifyNativeRMF - Syntax

Determines if the native rendering mechanism and format of an object match any supplied by the application.

```
#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGITEM    pDragitem; /* Pointer to the DRAGITEM structure. whose native rendering mechanism and format are
PSZ           pRMF;      /* A string specifying the rendering mechanism and format. */
BOOL         rc;         /* Validity indicator. */

rc = DrgVerifyNativeRMF(pDragitem, pRMF);
```

DrgVerifyNativeRMF Parameter - pDragitem

pDragitem (**PDRAGITEM**) - input
Pointer to the **DRAGITEM** structure. whose native rendering mechanism and format are to be verified.

DrgVerifyNativeRMF Parameter - pRMF

pRMF (**PSZ**) - input
A string specifying the rendering mechanism and format.

The string is of the form:

mechfmt[,mechfmt,mechfmt,...]

where mechfmt can be in either of these formats:

- <mechanism(1),format(1)>
- (mechanism(1)[,mechanism(n)...]), (format(1)[,format(n)...])

DrgVerifyNativeRMF Return Value - rc

rc (**BOOL**) - returns
Validity indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgVerifyNativeRMF - Parameters

pDragitem ([PDRAGITEM](#)) - input
Pointer to the [DRAGITEM](#) structure. whose native rendering mechanism and format are to be verified.

pRMF ([PSZ](#)) - input
A string specifying the rendering mechanism and format.

The string is of the form:

```
mechfmt [ ,mechfmt ,mechfmt , . . . ]
```

where mechfmt can be in either of these formats:

- <mechanism(1),format(1)>
- (mechanism(1)[,mechanism(n)...]), (format(1)[,format(n)...])

rc ([BOOL](#)) - returns
Validity indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgVerifyNativeRMF - Remarks

This function determines if the native rendering mechanism and format of a dragged object are understood by the target.

If TRUE is returned, the target may be able to carry out the action indicated by the direct manipulation itself, or it can select the native rendering mechanism and format as those to be used for the data exchange.

DrgVerifyNativeRMF - Related Functions

Related Functions

- [DrgQueryNativeRMF](#)
-

DrgVerifyNativeRMF - Example Code

This example determines if the native rendering mechanism and format of an object match any supplied by the application.

```

#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

DRAGITEM Dragitem;      /* DRAGITEM structure whose native      */
                        /* rendering mechanism and format are  */
                        /* to be verified                      */

char pszRMF[25];        /* A string specifying the rendering    */
                        /* mechanism and format. The string is */
                        /* of the form:                        */
                        /*                                     */
                        /* mechfmt[,mechfmt,mechfmt,...],      */
                        /*                                     */
                        /* where 'mechfmt' can be in either of */
                        /* these formats:                      */
                        /*                                     */
                        /* o <mechanism(1),format(1)>          */
                        /* o (mechanism(1)[,mechanism(n)...]), */
                        /*   (format(1)[,format(n)...])        */

strcpy(pszRMF,"<DRM_OS2FILE,DRF_TEXT>");
                        /* Mechanism is an OS/2 file and format */
                        /* is a null-terminated string. See   */
                        /* the DRAGITEM structure for valid   */
                        /* formats.                                */

if(DrgVerifyNativeRMF(&Dragitem, pszRMF))
{
    /* Code block
}

```

DrgVerifyNativeRMF - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgVerifyRMF

DrgVerifyRMF - Syntax

This function determines if a given rendering mechanism and format are supported for a dragged object.

```

#define INCL_WINSTDDRAG
#include <os2.h>

```

```
PDRAGITEM    pDragitem; /* Pointer to the DRAGITEM structure whose native rendering mechanism and format are
PSZ           pMech;    /* String specifying the rendering mechanism to search for. */
PSZ           pFormat;  /* String specifying the rendering format to search for. */
BOOL          rc;       /* Validity indicator. */

rc = DrgVerifyRMF(pDragitem, pMech, pFormat);
```

DrgVerifyRMF Parameter - pDragitem

pDragitem (**PDRAGITEM**) - input
Pointer to the **DRAGITEM** structure whose native rendering mechanism and format are to be validated.

DrgVerifyRMF Parameter - pMech

pMech (**PSZ**) - input
String specifying the rendering mechanism to search for.

NULL will match any mechanism.

DrgVerifyRMF Parameter - pFormat

pFormat (**PSZ**) - input
String specifying the rendering format to search for.

NULL will match any format.

DrgVerifyRMF Return Value - rc

rc (**BOOL**) - returns
Validity indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgVerifyRMF - Parameters

pDragitem ([PDRAGITEM](#)) - input
Pointer to the [DRAGITEM](#) structure whose native rendering mechanism and format are to be validated.

pMech ([PSZ](#)) - input
String specifying the rendering mechanism to search for.

NULL will match any mechanism.

pFormat ([PSZ](#)) - input
String specifying the rendering format to search for.

NULL will match any format.

rc ([BOOL](#)) - returns
Validity indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgVerifyRMF - Remarks

This function determines if a given rendering mechanism and format ordered pair are represented in the set of valid pairs specified by */strRMF* for the dragged object.

DrgVerifyRMF - Related Functions

Related Functions

- [DrgVerifyNativeRMF](#)

DrgVerifyRMF - Example Code

This example determines if a given rendering mechanism and format are supported for a dragged object.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <OS2.H>

DRAGITEM Dragitem;      /* DRAGITEM structure whose native      */
                        /* rendering mechanism and format are    */
                        /* to be validated                       */
char pszMech[] = "DRM_OS2FILE";
                        /* A string specifying the rendering    */
                        /* mechanism to search for             */
char pszFormat[] = "DRF_TEXT";
                        /* A string specifying the rendering    */
                        /* format to search for                 */

if(DrgVerifyRMF(&Dragitem, pszMech, pszFormat))
                        /* Mechanism is an OS/2 file and format */
                        /* is a null-terminated string      */
{
                        /* Code block                                     */
}
```

```
}
```

DrgVerifyRMF - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

DrgVerifyTrueType

DrgVerifyTrueType - Syntax

This function determines if the true type of a dragged object matches an application-supplied type string.

```
#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGITEM    pDragitem; /* Pointer to the DRAGITEM structure whose true type is to be verified. */
PSZ          pType;     /* String specifying a type. */
BOOL         rc;        /* Validity indicator. */

rc = DrgVerifyTrueType(pDragitem, pType);
```

DrgVerifyTrueType Parameter - pDragitem

pDragitem ([PDRAGITEM](#)) - input

Pointer to the [DRAGITEM](#) structure whose true type is to be verified.

DrgVerifyTrueType Parameter - pType

pType ([PSZ](#)) - input

String specifying a type.

This string is in the format:

```
type[ , type... ]
```

DrgVerifyTrueType Return Value - rc

rc (**BOOL**) - returns
Validity indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgVerifyTrueType - Parameters

pDragitem (**PDRAGITEM**) - input
Pointer to the **DRAGITEM** structure whose true type is to be verified.

pType (**PSZ**) - input
String specifying a type.

This string is in the format:

```
type[ , type... ]
```

rc (**BOOL**) - returns
Validity indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgVerifyTrueType - Remarks

If an item in the string pointed to by *pType* matches the first type in the string associated with *hstrType* in the **DRAGITEM** structure, TRUE is returned.

A target application uses this function to determine if it supports the true type of a dragged object. If the application does not support the true type, it can either disallow a drop or change its default operation. If the default operation is a move, the drop should be disallowed, or the operation changed to a copy to prevent any loss of data for the object.

DrgVerifyTrueType - Related Functions

Related Functions

- [DrgQueryTrueType](#)
- [DrgVerifyType](#)
- [DrgVerifyTypeSet](#)

DrgVerifyTrueType - Example Code

This example verifies whether a given type is present in the list of types defined for a drag object.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <OS2.H>

BOOL      fValid;
DRAGITEM  Dragitem;      /* DRAGITEM structure whose hstrType is */
                          /* to be verified */

char pszType[8];          /* A string specifying the types to */
                          /* search for */

strcpy(pszType,DRT_EXE); /* Executable file type. See the */
                          /* DRAGINFO structure for valid */
                          /* types. */

fValid = DrgVerifyTrueType(&Dragitem, pszType);
```

DrgVerifyTrueType - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgVerifyType

DrgVerifyType - Syntax

This function verifies whether a given type is present in the list of types defined for a drag object.

```

#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGITEM    pDragitem; /* Pointer to the DRAGITEM structure whose hstrType is to be verified. */
PSZ          pType;     /* String specifying the types to search for. */
BOOL         rc;        /* Success indicator. */

rc = DrgVerifyType(pDragitem, pType);

```

DrgVerifyType Parameter - pDragitem

pDragitem (**PDRAGITEM**) - input
 Pointer to the **DRAGITEM** structure whose *hstrType* is to be verified.

DrgVerifyType Parameter - pType

pType (**PSZ**) - input
 String specifying the types to search for.

This string is in the format:

`type[, type...]`

DrgVerifyType Return Value - rc

rc (**BOOL**) - returns
 Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgVerifyType - Parameters

pDragitem (**PDRAGITEM**) - input
 Pointer to the **DRAGITEM** structure whose *hstrType* is to be verified.

pType ([PSZ](#)) - input
String specifying the types to search for.

This string is in the format:

`type[, type...]`

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgVerifyType - Remarks

If at least one of the types specified by *pType* is present in *hstrType* in the [DRAGITEM](#) structure, TRUE is returned.

DrgVerifyType - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_PARAMETERS (0x1208)

An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32 768 to +32 767 cannot be converted to a [SHORT](#), and a negative number cannot be converted to a [ULONG](#) or [USHORT](#).

PMERR_INSUFFICIENT_MEMORY (0x203E)

The operation terminated through insufficient memory.

DrgVerifyType - Related Functions

Related Functions

- [DrgVerifyTrueType](#)
 - [DrgVerifyTypeSet](#)
-

DrgVerifyType - Example Code

This example verifies whether a given type is present in the list of types defined for a drag object.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <OS2.H>
```

```
BOOL    fValid;
```

```

DRAGITEM Dragitem;          /* DRAGITEM structure whose hstrType is */
                             /* to be verified */
char pszType[] = DRT_EXE;   /* A string specifying the types to */
                             /* search for */

fValid = DrgVerifyType(&Dragitem, pszType);

```

DrgVerifyType - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DrgVerifyTypeSet

DrgVerifyTypeSet - Syntax

This function returns the intersection of the contents of the string associated with the type-string handle for an object and an application-specified type string.

```

#define INCL_WINSTDDRAG
#include <os2.h>

PDRAGITEM    pDragitem; /* Pointer to the DRAGITEM structure whose hstrType is to be verified. */
PSZ          pType;     /* String specifying the types to search for. */
ULONG        cbBuflen;  /* Size of the return buffer. */
PSZ          pBuffer;   /* Buffer where the intersection string is returned. */
BOOL         rc;        /* Match indicator. */

rc = DrgVerifyTypeSet(pDragitem, pType, cbBuflen,
                     pBuffer);

```

DrgVerifyTypeSet Parameter - pDragitem

pDragitem ([PDRAGITEM](#)) - input
 Pointer to the [DRAGITEM](#) structure whose *hstrType* is to be verified.

DrgVerifyTypeSet Parameter - pType

pType (PSZ) - input
String specifying the types to search for.

This string is in the format:

type[, type...]

DrgVerifyTypeSet Parameter - cbBuflen

cbBuflen (ULONG) - input
Size of the return buffer.

The buffer should be at least one byte longer than the length of the string pointed to by *pType*. Must be > 0.

DrgVerifyTypeSet Parameter - pBuffer

pBuffer (PSZ) - output
Buffer where the intersection string is returned.

DrgVerifyTypeSet Return Value - rc

rc (BOOL) - returns
Match indicator.

TRUE Successful completion.
FALSE Error occurred.

DrgVerifyTypeSet - Parameters

pDragitem (PDRAGITEM) - input

Pointer to the [DRAGITEM](#) structure whose *hstrType* is to be verified.

pType ([PSZ](#)) - input
String specifying the types to search for.

This string is in the format:

```
type[ , type... ]
```

cbBuflen ([ULONG](#)) - input
Size of the return buffer.

The buffer should be at least one byte longer than the length of the string pointed to by *pType*. Must be > 0.

pBuffer ([PSZ](#)) - output
Buffer where the intersection string is returned.

rc ([BOOL](#)) - returns
Match indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DrgVerifyTypeSet - Remarks

If at least one of the types specified by *pType* is present in *hstrType* in the [DRAGITEM](#) structure, TRUE is returned.

DrgVerifyTypeSet - Related Functions

Related Functions

- [DrgVerifyType](#)
 - [DrgVerifyTrueType](#)
-

DrgVerifyTypeSet - Example Code

In this example, the DrgVerifyTypeSet function is used to determine whether DRT_TEXT is among the types associated with the object. If it is, the drop is accepted.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <OS2.H>
#include <stdio.h>
BOOL flResult;
DRAGITEM pditem;
char szBuffer[32];

case DM_DRAGOVER:

    flResult = DrgVerifyTypeSet(&pditem,
                                DRT_TEXT,
```

```

        sizeof(szBuffer),
        szBuffer);

flResult = strcmp(szBuffer,DRT_TEXT);

/*****
/* See if the object is an OS/2 file as well as being of text */
/* format. AND result flag with previous result flag to get */
/* the "effective" return code. */
*****/

flResult = DrgVerifyRMF(&pditem,"DRM_OS2FILE","DRF_TEXT");

/*****
/* See if DRT_TEXT is the true type of the object */
*****/

flResult = DrgVerifyTrueType(&pditem,"DRF_TEXT");

if(!flResult)

/*****
/* Inform the application that you will accept the drop */
*****/

return(MRFROM2SHORT(DOR_DROP, DO_COPY));
break;

```

DrgVerifyTypeSet - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Example Code](#)
 - [Related Functions](#)
 - [Glossary](#)

Dynamic Data Formatting Functions

The Information Presentation Facility (IPF) manages online, context-sensitive help information. Dynamic Data Formatting (DDF) is used for displaying dynamic help information. This section contains the DDF functions used to accomplish Dynamic Data Formatting.

DdfBeginList

DdfBeginList - Syntax

This function begins a definition list in the DDF buffer; it corresponds to the **:dl.** (definition list) tag.

```

#define INCL_DDF
#include <os2.h>

HDDF      hddf;          /* Handle to DDF returned by DdfInitialize. */
ULONG     ulWidthDT;     /* Width of the definition term. */
ULONG     fBreakType;    /* Type of line break to use. */
ULONG     fSpacing;      /* Spacing between definitions. */
BOOL      rc;            /* Success indicator. */

rc = DdfBeginList(hddf, ulWidthDT, fBreakType,
                  fSpacing);

```

DdfBeginList Parameter - hddf

hddf (**HDDF**) - input
Handle to DDF returned by [DdfInitialize](#).

DdfBeginList Parameter - ulWidthDT

ulWidthDT (**ULONG**) - input
Width of the definition term.

DdfBeginList Parameter - fBreakType

fBreakType (**ULONG**) - input
Type of line break to use.

The following constants may be specified:

HMBT_ALL	Start all definition descriptions on the next line, regardless of the actual lengths of definition terms.
HMBT_FIT	Start definition description on the next line only when the definition term is longer than the width specified.
HMBT_NONE	Do not start the definition description on the next line, even when the definition term is longer than the width specified.

DdfBeginList Parameter - fSpacing

fSpacing (**ULONG**) - input

Spacing between definitions.

Only the following constants may be specified:

HMLS_SINGLELINE

Do not insert a blank line between each definition description and the next definition term.

HMLS_DOUBLELINE

Insert a blank line between each definition description and the next definition term.

DdfBeginList Return Value - rc

rc (**BOOL**) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

DdfBeginList - Parameters

hddf (**HDDF**) - input

Handle to DDF returned by [DdfInitialize](#).

ulWidthDT (**ULONG**) - input

Width of the definition term.

fBreakType (**ULONG**) - input

Type of line break to use.

The following constants may be specified:

HMBT_ALL

Start all definition descriptions on the next line, regardless of the actual lengths of definition terms.

HMBT_FIT

Start definition description on the next line only when the definition term is longer than the width specified.

HMBT_NONE

Do not start the definition description on the next line, even when the definition term is longer than the width specified.

fSpacing (**ULONG**) - input

Spacing between definitions.

Only the following constants may be specified:

HMLS_SINGLELINE

Do not insert a blank line between each definition description and the next definition term.

HMLS_DOUBLELINE

Insert a blank line between each definition description and the next definition term.

rc (**BOOL**) - returns

Success indicator.

TRUE

FALSE	Successful completion
	Error occurred.

DdfBeginList - Remarks

Once this function has been called, use of any DDF function other than [DdfListItem](#), [DdfSetColor](#), and [DdfEndList](#) may produce unpredictable results.

DdfBeginList - Errors

Possible returns from [WinGetLastError](#)

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

HMERR_DDF_LIST_UNCLOSED (0x3007)
An attempt was made to nest a list.

HMERR_DDF_LIST_BREAKTYPE (0x3009)
The value of BreakType is not valid.

HMERR_DDF_LIST_SPACING (0x300A)
The value for Spacing is not valid.

DdfBeginList - Related Functions

Related Functions

- [DdfBeginList](#)
- [DdfBitmap](#)
- [DdfEndList](#)
- [DdfHyperText](#)
- [DdfInform](#)
- [DdfInitialize](#)
- [DdfListItem](#)
- [DdfMetafile](#)
- [DdfPara](#)
- [DdfSetColor](#)
- [DdfSetFont](#)
- [DdfSetFontStyle](#)
- [DdfSetFormat](#)
- [DdfSetTextAlign](#)
- [DdfText](#)

DdfBeginList - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses [DdfBeginList](#) to indicate the beginning of a definition list in the DDF buffer (this corresponds to the IPF dl tag). For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```

#define INCL_WINWINDOWMGR          /* General window management */
#define INCL_WINMESSAGEMGR        /* Message management */
#define INCL_DDF                   /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>

struct _LISTITEM                   /* definition list */
{
    PSZ Term;
    PSZ Desc;
} Definition[2] = {{ "MVS", "Multiple Virtual System"},
                  { "VM", "Virtual Machine"} };

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2)
{
    HWND  hwndParent;
    HWND  hwndInstance;
    Hddf  hDdf;           /* DDF handle */
    SHORT i;              /* loop index */

    switch( ulMsg )
    {
        case HM_QUERY_DDF_DATA:
            /* get the help instance */
            hwndParent = WinQueryWindow( hwnd, QW_PARENT );
            hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
            hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                             MPFROMSHORT( HMQW_INSTANCE ), NULL );

            /* Allocate 1K Buffer (default) */
            hDdf = DdfInitialize(
                hwndInstance, /* Handle of help instance */
                0L,          /* Default buffer size */
                0L,          /* Default increment */
            );

            if (hDdf == NULLHANDLE) /* Check return code */
            {
                return (MRESULT)FALSE;
            }

            /* begin definition list */
            if (!DdfBeginList(hDdf, 3L, HMBT_ALL, HMLS_SINGLELINE))
            {
                return (MRESULT)FALSE;
            }

            /* insert 2 entries into definition list */
            for (i=0; i < 2; i++)
            {
                if (!DdfListItem(hDdf, Definition[i].Term,
                                Definition[i].Desc))
                {
                    return (MRESULT)FALSE;
                }
            }

            /* terminate definition list */
            if (!DdfEndList(hDdf))
            {
                return (MRESULT)FALSE;
            }

            return (MRESULT)hDdf;
        }
    }
}

```

DdfBeginList - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

DdfBitmap

DdfBitmap - Syntax

This function places a reference to a bit map in the DDF buffer.

```
#define INCL_DDF
#include <os2.h>

HDDF      hddf;      /* Handle to DDF returned by DdfInitialize. */
HBITMAP    hbm;      /* Standard Presentation Manager bit map handle. */
ULONG      fAlign;    /* Text justification flag. */
BOOL       rc;        /* Success indicator. */

rc = DdfBitmap(hddf, hbm, fAlign);
```

DdfBitmap Parameter - hddf

hddf ([HDDF](#)) - input
Handle to DDF returned by [DdfInitialize](#).

DdfBitmap Parameter - hbm

hbm ([HBITMAP](#)) - input
Standard Presentation Manager bit map handle.

DdfBitmap Parameter - fAlign

fAlign (ULONG) - input
Text justification flag.

Any of the following values can be specified:

- | | |
|------------|---|
| ART_LEFT | Left-justify the bit map. |
| ART_RIGHT | Right-justify the bit map. |
| ART_CENTER | Center the bit map. |
| ART_RUNIN | Allow the bit map to be reflowed with text. |

DdfBitmap Return Value - rc

rc (BOOL) - returns
Success indicator.

- | | |
|-------|-----------------------|
| TRUE | Successful completion |
| FALSE | Error occurred. |

DdfBitmap - Parameters

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

hbm (HBITMAP) - input
Standard Presentation Manager bit map handle.

fAlign (ULONG) - input
Text justification flag.

Any of the following values can be specified:

- | | |
|------------|---|
| ART_LEFT | Left-justify the bit map. |
| ART_RIGHT | Right-justify the bit map. |
| ART_CENTER | Center the bit map. |
| ART_RUNIN | Allow the bit map to be reflowed with text. |

rc (BOOL) - returns
Success indicator.

- | | |
|-------|-----------------------|
| TRUE | Successful completion |
| FALSE | Error occurred. |

DdfBitmap - Remarks

The handle to the presentation space in which the bit map was created cannot be freed by the application while the panel is displayed.

Note: There is a (3-byte + size of [HBITMAP](#) structure). ESC code overhead in the DDF internal buffer for this function. There is a 1-byte ESC code overhead required for the *Align* flag.

DdfBitmap - Errors

Possible returns from [WinGetLastError](#)

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

HMERR_DDF_ALIGN_TYPE (0x3002)
The alignment type is not valid.

DdfBitmap - Related Functions

Related Functions

- [DdfBeginList](#)
 - [DdfBitmap](#)
 - [DdfEndList](#)
 - [DdfHyperText](#)
 - [DdfInform](#)
 - [DdfInitialize](#)
 - [DdfListItem](#)
 - [DdfMetafile](#)
 - [DdfPara](#)
 - [DdfSetColor](#)
 - [DdfSetFont](#)
 - [DdfSetFontStyle](#)
 - [DdfSetFormat](#)
 - [DdfSetTextAlign](#)
 - [DdfText](#)
-

DdfBitmap - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example obtains a device context (DevOpenDC), creates a presentation space (GpiCreatePS), and loads a bit map (GpiLoadBitmap). It then uses DdfBitmap to place a reference to the bit map in the DDF buffer. For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEGR     /* Message management */
#define INCL_GPICONTROL       /* Basic PS control */
#define INCL_GPIBITMAPS       /* Bit maps and Pel Operations */
#define INCL_GPIPRIMITIVES    /* Drawing Primitives/Attributes */
```

```

#define INCL_DDF                      /* Dynamic Data Facility          */
#include <os2.h>
#include <pmhelp.h>

#define ACVP_HAB  12
#define BM_HPS    16
#define BM_HDC    20
#define BM_HWND   24
#define ID_LEFT   255

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND    hwndParent;    /* parent window                */
    HWND    hwndInstance;  /* help instance window         */
    Hddf    hDdf;          /* DDF handle                   */
    HDC     hdc;           /* device context handle        */
    HPS     hps;           /* presentation space handle    */
    HAB     hab;           /* anchor block handle          */
    SIZEL    sizel = {0L,0L}; /* size of new PS              */
    HBITMAP hBitmap;       /* bit map handle               */
    HMODULE  hModule;       /* module handle                 */

    switch( ulMsg )
    {
        case HM_QUERY_DDF_DATA:
            hwndParent = WinQueryWindow( hwnd, QW_PARENT );
            hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
            hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                             MPFROMSHORT( HMQW_INSTANCE ), NULL );

            /* Allocate 1K Buffer (default) */
            hDdf = DdfInitialize(
                hwndInstance, /* Handle of help instance */
                0L,          /* Default buffer size     */
                0L           /* Default increment       */
            );

            if (hDdf == NULLHANDLE) /* Check return code */
            {
                return (MRESULT)FALSE;
            }

            /* get module handle for bit map */
            DosQueryModuleHandle("bitmap", &hModule);
            if (hModule == NULLHANDLE)
            {
                return (MRESULT)FALSE;
            }

            /* get hab for this window */
            if ((hab = (HAB)WinQueryWindowULong(hwnd, ACVP_HAB)) == NULLHANDLE)
            {
                return (MRESULT)FALSE;
            }

            /* create a device context */
            if ((hdc = DevOpenDC(hab, OD_MEMORY, "", 0L,
                                (PDEVOPENDATA)NULL, (HDC)NULL)) == NULLHANDLE)
            {
                return (MRESULT)FALSE;
            }

            /* save hdc in reserved word */
            WinSetWindowULong(hwnd, BM_HDC, (ULONG)hdc);

            /* create a noncached micro presentation space */
            /* and associate it with the window */
            if ((hps = GpiCreatePS(hab, hdc, &sizel,
                                   PU_PELS | GPIF_DEFAULT
                                   | GPIT_MICRO | GPIA_ASSOC)) == NULLHANDLE)
            {
                return (MRESULT)FALSE;
            }

            /* save hps in reserved word */
            WinSetWindowULong(hwnd, BM_HPS, (ULONG)hps);

            /* Load the Bit map to display */
            if ((hBitmap = GpiLoadBitmap(hps, hModule, ID_LEFT, 300L,

```

```

        300L)) == NULLHANDLE)
    {
        return (MRESULT)FALSE;
    }

    /* save bit map hwnd in reserved word */
    WinSetWindowULong(hwnd, BM_HWND, (ULONG)hBitmap);

    /* Display the bit map align left */
    if (!DdfBitmap(hDdf, hBitmap, ART_LEFT))
    {
        return (MRESULT)FALSE;
    }

    return (MRESULT)hDdf;

case WM_CLOSE:
    /* release PS, DC, and bit map */
    GpiDestroyPS((HPS)WinQueryWindowULong(hwnd, BM_HPS));
    DevCloseDC((HDC)WinQueryWindowULong(hwnd, BM_HDC));
    GpiDeleteBitmap((HBITMAP)WinQueryWindowULong(hwnd, BM_HWND));
    WinDestroyWindow(WinQueryWindow(hwnd, QW_PARENT));
    return (MRESULT)TRUE;
}
}

```

DdfBitmap - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DdfEndList

DdfEndList - Syntax

This function terminates the definition list initialized by DdfBeginList.

```

#define INCL_DDF
#include <os2.h>

HDDF    hddf; /* Handle to DDF returned by DdfInitialize. */
BOOL    rc;    /* Success indicator. */

rc = DdfEndList(hddf);

```

DdfEndList Parameter - hddf

hddf ([HDDF](#)) - input
Handle to DDF returned by [DdfInitialize](#).

DdfEndList Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfEndList - Parameters

hddf ([HDDF](#)) - input
Handle to DDF returned by [DdfInitialize](#).

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfEndList - Errors

Possible returns from [WinGetLastError](#)

HMERR_DDF_LIST_UNINITIALIZED (0x3008)
No definition list has been initialized by [DdfBeginList](#).

DdfEndList - Related Functions

Related Functions

- [DdfBeginList](#)
- [DdfBitmap](#)
- [DdfEndList](#)
- [DdfHyperText](#)
- [DdfInform](#)
- [DdfInitialize](#)
- [DdfListItem](#)
- [DdfMetafile](#)
- [DdfPara](#)
- [DdfSetColor](#)
- [DdfSetFont](#)
- [DdfSetFontStyle](#)
- [DdfSetFormat](#)
- [DdfSetTextAlign](#)
- [DdfText](#)

DdfEndList - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses [DdfBeginList](#) to indicate the beginning of a definition list in the DDF buffer (this corresponds to the IPF dl tag). For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEGR     /* Message management */
#define INCL_DDF              /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>

struct _LISTITEM              /* definition list */
{
    PSZ Term;
    PSZ Desc;
} Definition[2] = {{"MVS", "Multiple Virtual System"},
                  {"VM", "Virtual Machine"}};

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2)
{
    HWND    hwndParent;
    HWND    hwndInstance;
    Hddf    hDdf;          /* DDF handle */
    SHORT   i;             /* loop index */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                         MPFROMSHORT( HMQW_INSTANCE ), NULL );

        /* Allocate 1K Buffer (default) */
        hDdf = DdfInitialize(
            hwndInstance, /* Handle of help instance */
            0L,          /* Default buffer size */
            0L,          /* Default increment */
        );

        if (hDdf == NULLHANDLE) /* Check return code */
        {
            return (MRESULT)FALSE;
        }

        /* begin definition list */
        if (!DdfBeginList(hDdf, 3L, HMBT_ALL, HMLS_SINGLELINE))
        {
            return (MRESULT)FALSE;
        }

        /* insert 2 entries into definition list */
    }
```

```

    for (i=0; i < 2; i++)
    {
        if (!DdfListItem(hDdf, Definition[i].Term,
                        Definition[i].Desc))
        {
            return (MRESULT)FALSE;
        }
    }

    /* terminate definition list */
    if (!DdfEndList(hDdf))
    {
        return (MRESULT)FALSE;
    }

    return (MRESULT)hDdf;
}
}

```

DdfEndList - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

DdfHyperText

DdfHyperText - Syntax

This function defines a hypertext link to another panel, which is equal to a link of reftype=hd. Links to footnotes, launch links and links to external databases are not supported.

```

#define INCL_DDF
#include <os2.h>

HDDF      hddf;          /* Handle to DDF returned by DdfInitialize. */
PCSZ      pszText;        /* Hypertext phrase. */
PCSZ      pszReference;    /* Pointer to a string containing the link reference. */
ULONG     fReferenceType; /* Reference Type. */
BOOL      rc;             /* Success indicator. */

rc = DdfHyperText(hddf, pszText, pszReference,
                  fReferenceType);

```

DdfHyperText Parameter - hddf

hddf (**HDDF**) - input
Handle to DDF returned by [DdfInitialize](#).

DdfHyperText Parameter - pszText

pszText (**PCSZ**) - input
Hypertext phrase.

DdfHyperText Parameter - pszReference

pszReference (**PCSZ**) - input
Pointer to a string containing the link reference.

The value of this parameter depends on the value of *fReferenceType*.

If *fReferenceType* is **REFERENCE_BY_RES**
pszReference must contain a pointer to a numeric string containing the res number; otherwise it will default to a res number of zero. Valid values are 1 - 64000; all other values are reserved.

If *fReferenceType* is **REFERENCE_BY_ID**
pszReference must contain a pointer to a string containing the alphanumeric identifier of the destination panel.

DdfHyperText Parameter - fReferenceType

fReferenceType (**ULONG**) - input
Reference Type.

This parameter specifies whether you are linking via a resource identifier (res number) or via an alphanumeric identifier.

REFERENCE_BY_RES
To link via a resource identifier.

REFERENCE_BY_ID
To link via an alphanumeric identifier.

DdfHyperText Return Value - rc

rc (**BOOL**) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

DdfHyperText - Parameters

hddf ([HDDF](#)) - input

Handle to DDF returned by [DdfInitialize](#).

pszText ([PCSZ](#)) - input

Hypertext phrase.

pszReference ([PCSZ](#)) - input

Pointer to a string containing the link reference.

The value of this parameter depends on the value of *fReferenceType*.

If *fReferenceType* is `REFERENCE_BY_RES`

pszReference must contain a pointer to a numeric string containing the res number; otherwise it will default to a res number of zero. Valid values are 1 - 64000; all other values are reserved.

If *fReferenceType* is `REFERENCE_BY_ID`

pszReference must contain a pointer to a string containing the alphanumeric identifier of the destination panel.

fReferenceType ([ULONG](#)) - input

Reference Type.

This parameter specifies whether you are linking via a resource identifier (res number) or via an alphanumeric identifier.

`REFERENCE_BY_RES`

To link via a resource identifier.

`REFERENCE_BY_ID`

To link via an alphanumeric identifier.

rc ([BOOL](#)) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

DdfHyperText - Remarks

There is a 3-byte ESC code overhead in the DDF internal buffer for each word in the text buffer. There is a 1-byte ESC code overhead for each blank and for each newline character. If *fReferenceType* is `REFERENCE_BY_ID`, then there is a (3-byte + Reference length) ESC code overhead. For a *fReferenceType* of `REFERENCE_BY_RES`, the overhead is 5 bytes. Finally, there is a 3-byte ESC code overhead that is required for ending the hypertext link.

DdfHyperText - Errors

Possible returns from [WinGetLastError](#)

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

HMERR_DDF_REFTYPE (0x3006)
The reference type is not valid.

DdfHyperText - Related Functions

Related Functions

- [DdfBeginList](#)
- [DdfBitmap](#)
- [DdfEndList](#)
- [DdfHyperText](#)
- [DdfInform](#)
- [DdfInitialize](#)
- [DdfListItem](#)
- [DdfMetafile](#)
- [DdfPara](#)
- [DdfSetColor](#)
- [DdfSetFont](#)
- [DdfSetFontStyle](#)
- [DdfSetFormat](#)
- [DdfSetTextAlign](#)
- [DdfText](#)

DdfHyperText - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses DdfHyperText to create a hypertext link with another resource. For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEMGR    /* Message management */
#define INCL_DDF               /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>

PSZ    Text = "This text is a HYPERTEXT message.\n"; /* hypertext string */
PSZ    ResID = "1"; /* Resource identifier */

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND    hwndParent;
    HWND    hwndInstance;
    Hddf     hDdf; /* DDF handle */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                          MPFROMSHORT( HMQW_INSTANCE ), NULL );

        /* Allocate 1K Buffer (default) */
        hDdf = DdfInitialize(
            hwndInstance, /* Handle of help instance */
            0L,           /* Default buffer size */
            0L,           /* Default increment */
        );
    }
```

```

        if (hDdf == NULLHANDLE)          /* Check return code          */
        {
            return (MRESULT)FALSE;
        }

        /* create hypertext link with resource 1 */
        if (!DdfHyperText(hDdf, (PSZ)Text, ResID, REFERENCE_BY_RES))
        {
            return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
}

```

DdfHyperText - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DdfInform

DdfInform - Syntax

This function defines a hypertext inform link, which sends an HM_INFORM message to the application with a **res=** attribute. It also corresponds to the link tag with reftype=inform.

```

#define INCL_DDF
#include <os2.h>

HDDF      hddf;          /* Handle to DDF returned by DdfInitialize. */
PCSZ      pszText;       /* Hypertext phrase. */
ULONG     resInformNumber; /* Res number associated with this hypertext field. */
BOOL      rc;            /* Success indicator. */

rc = DdfInform(hddf, pszText, resInformNumber);

```

DdfInform Parameter - hddf

hddf ([HDDF](#)) - input
Handle to DDF returned by [DdfInitialize](#).

DdfInform Parameter - pszText

pszText ([PCSZ](#)) - input
Hypertext phrase.

DdfInform Parameter - resInformNumber

resInformNumber ([ULONG](#)) - input
Res number associated with this hypertext field.

Possible values are 1 to 64000; all other values are reserved.

DdfInform Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfInform - Parameters

hddf ([HDDF](#)) - input
Handle to DDF returned by [DdfInitialize](#).

pszText ([PCSZ](#)) - input
Hypertext phrase.

resInformNumber ([ULONG](#)) - input
Res number associated with this hypertext field.

Possible values are 1 to 64000; all other values are reserved.

rc ([BOOL](#)) - returns
Success indicator.

TRUE

FALSE Successful completion
 Error occurred.

DdfInform - Errors

Possible returns from [WinGetLastError](#)

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

DdfInform - Related Functions

Related Functions

- [DdfBeginList](#)
 - [DdfBitmap](#)
 - [DdfEndList](#)
 - [DdfHyperText](#)
 - [DdfInform](#)
 - [DdfInitialize](#)
 - [DdfListItem](#)
 - [DdfMetafile](#)
 - [DdfPara](#)
 - [DdfSetColor](#)
 - [DdfSetFont](#)
 - [DdfSetFontStyle](#)
 - [DdfSetFormat](#)
 - [DdfSetTextAlign](#)
 - [DdfText](#)
-

DdfInform - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses [DdfInform](#) to create a hypertext inform link with another resource (corresponds to the IPF :link. tag with reftype=inform). For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR            /* General window management    */
#define INCL_WINMESSAGEGR           /* Message management           */
#define INCL_DDF                    /* Dynamic Data Facility        */
#include <os2.h>
#include <pmhelp.h>

PSZ     Text = "This text is a HYPERTEXT message.\n"; /* hypertext string */
MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND    hwndParent;
    HWND    hwndInstance;
    HDDF    hDdf;                   /* DDF handle                   */

    switch( ulMsg )
    {
        case HM_QUERY_DDF_DATA:
            /* get the help instance */
            hwndParent = WinQueryWindow( hwnd, QW_PARENT );
```

```

        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                          MPFROMSHORT( HMQW_INSTANCE ), NULL );

/* Allocate 1K Buffer (default) */
hDdf = DdfInitialize(
        hwndInstance, /* Handle of help instance */
        0L,           /* Default buffer size */
        0L,           /* Default increment */
        );

if (hDdf == NULLHANDLE) /* Check return code */
{
    return (MRESULT)FALSE;
}

/* create hypertext inform link with resource 1 */
if (!DdfInform(hDdf, (PSZ)Text, 1L))
{
    return (MRESULT)FALSE;
}

return (MRESULT)hDdf;
}
}

```

DdfInform - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

DdfInitialize

DdfInitialize - Syntax

This function initializes the IPF internal structures for dynamic data formatting and returns a DDF handle. The application uses this handle to refer to a particular DDF panel.

```

#define INCL_DDF
#include <os2.h>

HWND    hwndHelpInstance; /* Handle of a help instance. */
ULONG   cbBuffer;         /* Initial length of internal buffer. */
ULONG   ulIncrement;      /* Amount by which to increment the buffer size, if necessary. */
HDDF    hddf;             /* Handle to DDF. */

hddf = DdfInitialize(hwndHelpInstance, cbBuffer,
                     ulIncrement);

```

DdfInitialize Parameter - hwndHelpInstance

hwndHelpInstance ([HWND](#)) - input
Handle of a help instance.

DdfInitialize Parameter - cbBuffer

cbBuffer ([ULONG](#)) - input
Initial length of internal buffer.

Initial length of internal buffer where DDF information is to be stored. If this field is 0L, a default value of 1K is defined. The maximum value is 61 440 bytes (60KB).

DdfInitialize Parameter - ullIncrement

ullIncrement ([ULONG](#)) - input
Amount by which to increment the buffer size, if necessary.

If this field is NULL, a default value of 256 bytes is defined. The maximum value is 60KB.

DdfInitialize Return Value - hddf

hddf ([HDDF](#)) - returns
Handle to DDF.

A handle to DDF is returned if initialization was successful. Otherwise, the value returned is NULL, indicating that an error has occurred because of insufficient memory or incorrect instance.

DdfInitialize - Parameters

hwndHelpInstance ([HWND](#)) - input
Handle of a help instance.

cbBuffer ([ULONG](#)) - input

Initial length of internal buffer.

Initial length of internal buffer where DDF information is to be stored. If this field is 0L, a default value of 1K is defined. The maximum value is 61 440 bytes (60KB).

ullIncrement ([ULONG](#)) - input

Amount by which to increment the buffer size, if necessary.

If this field is NULL, a default value of 256 bytes is defined. The maximum value is 60KB.

hddf ([HDDF](#)) - returns

Handle to DDF.

A handle to DDF is returned if initialization was successful. Otherwise, the value returned is NULL, indicating that an error has occurred because of insufficient memory or incorrect instance.

DdfInitialize - Remarks

At initialization, the default for dynamic data display is that text aligned on the left, and formatting is turned on.

DdfInitialize - Related Functions

Related Functions

- [DdfBeginList](#)
- [DdfBitmap](#)
- [DdfEndList](#)
- [DdfHyperText](#)
- [DdfInform](#)
- [DdfInitialize](#)
- [DdfListItem](#)
- [DdfMetafile](#)
- [DdfPara](#)
- [DdfSetColor](#)
- [DdfSetFont](#)
- [DdfSetFontStyle](#)
- [DdfSetFormat](#)
- [DdfSetTextAlign](#)
- [DdfText](#)

DdfInitialize - Example Code

This example shows how to initialize and use the Dynamic Data Facility for displaying an online document. Two functions are defined: the first, `SampleObj`, creates a window that will display the online information and specifies the second function, `SampleWindowProc`, as the corresponding window procedure. These two functions are compiled into a DLL and exported, so that IPF can invoke them when it encounters the `:ddf` and `:acviewport` tags during execution. The `:acviewport` tag will specify the DLL name and the `SampleObj` function; when IPF calls `SampleObj`, it initializes an application-controlled window with `SampleWindowProc` as the window procedure and returns the window handle. Later, when IPF encounters the `:ddf` tag, it will send `SampleWindowProc` an `HM_QUERY_DDF_DATA` message. At this point, before calling any of the DDF API, `DdfInitialize` must first be called to initiate a DDF buffer, after which the other DDF API can be called to display the online information.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEGR     /* Message management */
#define INCL_WINDIALOGS       /* Dialog boxes */
#define INCL_DDF              /* Dynamic Data Facility */
```

```

#define INCL_32
#include <os2.h>
#include <pmhelp.h>

#define COM_HWND 4          /* window word offsets */
#define PAGE_HWND 8
#define ACVP_HAB 12

USHORT DdfClass = FALSE;

MRESULT EXPENTRY SampleWindowProc(HWND hWnd, ULONG Message,
                                   MPARAM lParam1, MPARAM lParam2);

USHORT APIENTRY SampleObj(PACVP pACVP, PCH Parameter)
{
    HWND DdfHwnd;          /* Client window handle */
    HWND DdfCHwnd;         /* Child window handle */
    HWND PreviousHwnd;     /* Handle for setting comm window active */

    /* register DDF Base class if not registered already */
    if (!DdfClass)
    {
        if (!WinRegisterClass(
            pACVP->hAB,      /* Anchor block handle */
            "CLASS_Ddf",    /* Application window class name */
            SampleWindowProc, /* Address of window procedure */
            CS_SYNCPAINT |  /* Window class style */
            CS_SIZEREDRAW |
            CS_MOVENOTIFY,
            20))            /* Extra storage */
        {
            return TRUE;
        }
        DdfClass = TRUE;
    }

    /* create standard window */
    if (!(DdfHwnd = WinCreateStdWindow(
        pACVP->hWndParent, /* ACVP is parent */
        0L,               /* No class style */
        NULL,             /* Frame control flag */
        "CLASS_Ddf",      /* Window class name */
        NULL,             /* No title bar */
        0L,              /* No special style */
        0L,              /* Resource in .EXE */
        0,               /* No window identifier */
        &DdfCHwnd )))     /* Client window handle */
    {
        return FALSE;
    }

    /* store the frame window handle in ACVP data structure */
    pACVP->hWndACVP = DdfHwnd;

    /* set this window as active communication window */
    PreviousHwnd = (HWND)WinSendMsg(pACVP->hWndParent,
        HM_SET_OBJCOM_WINDOW,
        MPFROMHWNDDDF(DdfHwnd), NULL);

    /* save returned communication hwnd in reserved word */
    WinSetWindowULong(DdfCHwnd, COM_HWND, (ULONG)PreviousHwnd);

    /* save anchor block handle in reserved word */
    WinSetWindowULong(DdfCHwnd, ACVP_HAB, (ULONG)pACVP->hAB);

    return FALSE;
} /* SampleObj */

MRESULT EXPENTRY SampleWindowProc(HWND hWnd, ULONG Message,
                                   MPARAM lParam1, MPARAM lParam2)
{
    HWND hWndParent;       /* parent window */
    HWND hWndHelpInstance; /* help instance window */
    HDDF hDdf;            /* DDF handle */
    ULONG DdfID;           /* DDF resource id */

    switch (Message)
    {
        case HM_QUERY_DDF_DATA:

```



```

WinSetWindowULong(hWnd, PAGE_HWND, LONGFROMMP(lParam1));
DdfID = LONGFROMMP(lParam2);
hwndParent = WinQueryWindow(hWnd, QW_PARENT);
hwndParent = WinQueryWindow(hwndParent, QW_PARENT);
hwndHelpInstance = (HWND)WinSendMsg(hwndParent, HM_QUERY,
                                     MPFROMSHORT(HMQW_INSTANCE), NULL);

/* Allocate 1K Buffer (default) */
hDdf = DdfInitialize(
    hwndHelpInstance, /* Handle of help instance */
    0L,               /* Default buffer size */
    0L                /* Default increment */
);

if (hDdf == NULLHANDLE) /* Check return code */
{
    return (MRESULT)FALSE;
}

return (MRESULT)hDdf;

default:
    return (WinDefWindowProc(hWnd, Message, lParam1, lParam2));
} /* SampleWindowProc */

```

DdfInitialize - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DdfListItem

DdfListItem - Syntax

This function inserts a definition list entry in the DDF buffer; it corresponds to a combination of the **:dt.** (definition term) and **:dd.** (definition define) tags.

```

#define INCL_DDF
#include <os2.h>

HDDF    hddf;           /* Handle to DDF returned by DdfInitialize. */
PCSZ    pszTerm;        /* Term portion of the definition list entry. */
PCSZ    pszDescription; /* Description portion of the definition list entry. */
BOOL    rc;             /* Success indicator. */

rc = DdfListItem(hddf, pszTerm, pszDescription);

```

DdfListItem Parameter - hddf

hddf ([HDDF](#)) - input
Handle to DDF returned by [DdfInitialize](#).

DdfListItem Parameter - pszTerm

pszTerm ([PCSZ](#)) - input
Term portion of the definition list entry.

DdfListItem Parameter - pszDescription

pszDescription ([PCSZ](#)) - input
Description portion of the definition list entry.

Note: There is a 3-byte ESC code overhead in the DDF internal buffer for each word in both the term and the description. There is a 1-byte ESC code overhead for each blank and for each new-line character. For each list item, there is a 5-byte ESC code overhead for the margin alignment.

DdfListItem Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfListItem - Parameters

hddf ([HDDF](#)) - input
Handle to DDF returned by [DdfInitialize](#).

pszTerm ([PCSZ](#)) - input
Term portion of the definition list entry.

pszDescription ([PCSZ](#)) - input
Description portion of the definition list entry.

Note: There is a 3-byte ESC code overhead in the DDF internal buffer for each word in both the term and the description. There is a 1-byte ESC code overhead for each blank and for each new-line character. For each list item, there is a 5-byte ESC code overhead for the margin alignment.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfListItem - Remarks

The handle to the presentation space in which the bit map was created cannot be freed by the application while the panel is displayed.

Note: There is a (3-byte + size of HBITMAP structure) ESC code overhead in the DDF internal buffer for this function. There is a 1-byte ESC code overhead required for the Align flag.

DdfListItem - Errors

Possible returns from [WinGetLastError](#)

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

HMERR_DDF_LIST_UNINITIALIZED (0x3008)
No definition list has been initialized by [DdfBeginList](#).

DdfListItem - Related Functions

Related Functions

- [DdfBeginList](#)
- [DdfBitmap](#)
- [DdfEndList](#)
- [DdfHyperText](#)
- [DdfInform](#)
- [DdfInitialize](#)
- [DdfListItem](#)
- [DdfMetafile](#)
- [DdfPara](#)
- [DdfSetColor](#)
- [DdfSetFont](#)
- [DdfSetFontStyle](#)
- [DdfSetFormat](#)
- [DdfSetTextAlign](#)
- [DdfText](#)

DdfListItem - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses DdfBeginList to indicate the beginning of a definition list in the DDF buffer (this corresponds to the IPF dl tag). For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEGR     /* Message management */
#define INCL_DDF              /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>

struct _LISTITEM              /* definition list */
{
    PSZ Term;
    PSZ Desc;
} Definition[2] = {{ "MVS", "Multiple Virtual System"},
                  { "VM", "Virtual Machine"} };

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2)
{
    HWND    hwndParent;
    HWND    hwndInstance;
    Hddf     hDdf;
    SHORT i;

    /* DDF handle */
    /* loop index */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                         MPFROMSHORT( HMQW_INSTANCE ), NULL );

        /* Allocate 1K Buffer (default) */
        hDdf = DdfInitialize(
            hwndInstance, /* Handle of help instance */
            0L,           /* Default buffer size */
            0L,           /* Default increment */
        );

        if (hDdf == NULLHANDLE) /* Check return code */
        {
            return (MRESULT)FALSE;
        }

        /* begin definition list */
        if (!DdfBeginList(hDdf, 3L, HMBT_ALL, HMLS_SINGLELINE))
        {
            return (MRESULT)FALSE;
        }

        /* insert 2 entries into definition list */
        for (i=0; i < 2; i++)
        {
            if (!DdfListItem(hDdf, Definition[i].Term,
                             Definition[i].Desc))
            {
                return (MRESULT)FALSE;
            }
        }

        /* terminate definition list */
        if (!DdfEndList(hDdf))
        {
            return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
```

```
}  
}
```

DdfListItem - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

DdfMetafile

DdfMetafile - Syntax

This function places a reference to a metafile into the DDF buffer.

```
#define INCL_DDF  
#include <os2.h>  
  
HDDF      hddf;      /* Handle to DDF returned by DdfInitialize. */  
HMF      hmf;        /* The handle of the metafile to display. */  
PRECTL    prclRect;   /* Size of the rectangle. */  
BOOL      rc;         /* Success indicator. */  
  
rc = DdfMetafile(hddf, hmf, prclRect);
```

DdfMetafile Parameter - hddf

hddf (**HDDF**) - input

Handle to DDF returned by [DdfInitialize](#).

DdfMetafile Parameter - hmf

hmf ([HMF](#)) - input
The handle of the metafile to display.

DdfMetafile Parameter - prclRect

prclRect ([PRECTL](#)) - input
Size of the rectangle.

If not NULL, contains the size of the rectangle in which the metafile will be displayed. The aspect ratio of the metafile is adjusted to fit this rectangle.

DdfMetafile Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfMetafile - Parameters

hddf ([HDDF](#)) - input
Handle to DDF returned by [DdfInitialize](#).

hmf ([HMF](#)) - input
The handle of the metafile to display.

prclRect ([PRECTL](#)) - input
Size of the rectangle.

If not NULL, contains the size of the rectangle in which the metafile will be displayed. The aspect ratio of the metafile is adjusted to fit this rectangle.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfMetafile - Remarks

There is a 3-byte ESC code overhead in the DDF internal buffer for this function. There is also a (MetaFilename length) overhead. Finally, the *prcRect* variable requires an additional 16 bytes of overhead in the DDF internal buffer.

DdfMetafile - Errors

Possible returns from [WinGetLastError](#)

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

DdfMetafile - Related Functions

Related Functions

- [DdfBeginList](#)
 - [DdfBitmap](#)
 - [DdfEndList](#)
 - [DdfHyperText](#)
 - [DdfInform](#)
 - [DdfInitialize](#)
 - [DdfListItem](#)
 - [DdfMetafile](#)
 - [DdfPara](#)
 - [DdfSetColor](#)
 - [DdfSetFont](#)
 - [DdfSetFontStyle](#)
 - [DdfSetFormat](#)
 - [DdfSetTextAlign](#)
 - [DdfText](#)
-

DdfMetafile - Example Code

After initializing a DDF buffer with [DdfInitialize](#), and loading a metafile with [GpiLoadMetaFile](#), the example uses [DdfMetafile](#) to place a reference to the metafile in the DDF buffer. For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEGR     /* Message management */
#define INCL_DDF              /* Dynamic Data Facility */
#define INCL_GPIMETAFILES     /* MetaFiles */
#include <os2.h>
#include <pmhelp.h>

#define MF_HWND      0
#define ACVP_HAB     4

HRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND  hwndParent;
    HAB   hab;
    HWND  hwndInstance; /* help instance window */
    Hddf  hDdf;          /* DDF handle */
    HMF   hwndMetaFile;  /* metafile handle */

    switch( ulMsg )
```

```

{
case HM_QUERY_DDF_DATA:
    /* get the help instance */
    hwndParent = WinQueryWindow( hwnd, QW_PARENT );
    hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
    hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
        MPFROMSHORT( HMQW_INSTANCE ), NULL );

    /* Allocate 1K Buffer (default) */
    hDdf = DdfInitialize(
        hwndInstance, /* Handle of help instance */
        0L,           /* Default buffer size */
        0L,           /* Default increment */
        );

    if (hDdf == NULLHANDLE) /* Check return code */
    {
        return (MRESULT)FALSE;
    }

    /* get hab for this window */
    if ((hab = (HAB)WinQueryWindowULong(hwnd, ACVP_HAB)) == NULLHANDLE)
    {
        return (MRESULT)FALSE;
    }

    /* Load the Metafile to display */
    if ((hwndMetaFile = GpiLoadMetaFile(hab, "SAMP.MET")) == NULLHANDLE)
    {
        return (MRESULT)FALSE;
    }

    /* Save MetaFile hwnd in reserved word */
    WinSetWindowULong(hwnd, MF_HWND, hwndMetaFile);

    if (!DdfMetafile(hDdf, hwndMetaFile, NULL))
    {
        return (MRESULT)FALSE;
    }

    return (hDdf);

case WM_CLOSE:
    GpiDeleteMetaFile((HMF)WinQueryWindowULong(hwnd, MF_HWND));
    WinDestroyWindow(WinQueryWindow(hwnd, QW_PARENT));

    return (MRESULT)TRUE;
}
return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}

```

DdfMetafile - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

DdfPara

DdfPara - Syntax

This function creates a paragraph within the DDF buffer. It corresponds to the **:p.** tag. This function places a reference to a bit map in the DDF buffer.

```
#define INCL_DDF
#include <os2.h>

HDDF    hddf; /* Handle to DDF returned by DdfInitialize. */
BOOL    rc;    /* Success indicator. */

rc = DdfPara(hddf);
```

DdfPara Parameter - hddf

hddf (**HDDF**) - input
Handle to DDF returned by [DdfInitialize](#).

DdfPara Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfPara - Parameters

hddf (**HDDF**) - input
Handle to DDF returned by [DdfInitialize](#).

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfPara - Remarks

There is a 1-byte ESC code overhead in the DDF internal buffer for this function.

DdfPara - Errors

Possible returns from [WinGetLastError](#)

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

DdfPara - Related Functions

Related Functions

- [DdfBeginList](#)
 - [DdfBitmap](#)
 - [DdfEndList](#)
 - [DdfHyperText](#)
 - [DdfInform](#)
 - [DdfInitialize](#)
 - [DdfListItem](#)
 - [DdfMetafile](#)
 - [DdfPara](#)
 - [DdfSetColor](#)
 - [DdfSetFont](#)
 - [DdfSetFontStyle](#)
 - [DdfSetFormat](#)
 - [DdfSetTextAlign](#)
 - [DdfText](#)
-

DdfPara - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses [DdfPara](#) to start a new paragraph, [DdfSetFont](#) and [DdfSetFontStyle](#) to have the text displayed in a large, bold Courier font, [DdfSetColor](#) to change the text color, and [DdfText](#) to place text in the buffer. For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEGR     /* Message management */
#define INCL_DDF              /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND    hwndParent;
    HWND    hwndInstance;    /* help instance window */
    Hddf     hDdf;           /* DDF handle */
```

```

switch( ulMsg )
{
case HM_QUERY_DDF_DATA:
    /* get the help instance */
    hwndParent = WinQueryWindow( hwnd, QW_PARENT );
    hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
    hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                     MPFROMSHORT( HMQW_INSTANCE ), NULL );

    /* Allocate 1K Buffer (default) */
    hDdf = DdfInitialize(
        hwndInstance, /* Handle of help instance */
        0L,           /* Default buffer size */
        0L,           /* Default increment */
    );

    if (hDdf == NULLHANDLE) /* Check return code */
    {
        return (MRESULT)FALSE;
    }

    /* create paragraph in DDF buffer */
    if( !DdfPara( hDdf ) )
    {
        return (MRESULT)FALSE;
    }

    /* Change to large (100 x 100 dimensions) Courier font */
    if( !DdfSetFont( hDdf, "Courier", 100L, 100L ) )
    {
        return (MRESULT)FALSE;
    }

    /* make the font BOLDFACE */
    if( !DdfSetFontStyle( hDdf, FM_SEL_BOLD ) )
    {
        return (MRESULT)FALSE;
    }

    /* make the text display as BLUE on a PALE GRAY background */
    if( !DdfSetColor( hDdf, CLR_PALEGRAY, CLR_BLUE ) )
    {
        return (MRESULT)FALSE;
    }

    /* Write data into the buffer */
    if ( !DdfText(hDdf, "Sample Text") )
    {
        return (MRESULT)FALSE;
    }

    return (MRESULT)hDdf;
}
return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}

```

DdfPara - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DdfSetColor

DdfSetColor - Syntax

This function sets the background and foreground colors of the displayed text.

```
#define INCL_DDF
#include <os2.h>

HDDF      hddf;          /* Handle to DDF returned by DdfInitialize. */
COLOR     fBackColor;    /* Specifies the desired background color. */
COLOR     fForeColor;    /* Specifies the desired foreground color. */
BOOL      rc;            /* Success indicator. */

rc = DdfSetColor(hddf, fBackColor, fForeColor);
```

DdfSetColor Parameter - hddf

hddf ([HDDF](#)) - input
Handle to DDF returned by [DdfInitialize](#).

DdfSetColor Parameter - fBackColor

fBackColor ([COLOR](#)) - input
Specifies the desired background color.

DdfSetColor Parameter - fForeColor

fForeColor ([COLOR](#)) - input
Specifies the desired foreground color.

The following color value constants may be used for the foreground and background colors:

- CLR_DEFAULT - used to set IPF default text color
- CLR_BLACK
- CLR_BLUE
- CLR_RED
- CLR_PINK

CLR_GREEN
CLR_CYAN
CLR_YELLOW
CLR_BROWN
CLR_DARKGRAY
CLR_DARKBLUE
CLR_DARKRED
CLR_DARKPINK
CLR_DARKGREEN
CLR_DARKCYAN
CLR_PALEGGRAY

DdfSetColor Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfSetColor - Parameters

hddf (**HDDF**) - input
Handle to DDF returned by [DdfInitialize](#).

fBackColor (**COLOR**) - input
Specifies the desired background color.

fForeColor (**COLOR**) - input
Specifies the desired foreground color.

The following color value constants may be used for the foreground and background colors:

CLR_DEFAULT - used to set IPF default text color
CLR_BLACK
CLR_BLUE
CLR_RED
CLR_PINK
CLR_GREEN
CLR_CYAN
CLR_YELLOW
CLR_BROWN
CLR_DARKGRAY
CLR_DARKBLUE
CLR_DARKRED
CLR_DARKPINK
CLR_DARKGREEN
CLR_DARKCYAN
CLR_PALEGGRAY

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	

Error occurred.

DdfSetColor - Remarks

There is a 4-byte ESC code overhead in the DDF internal buffer for the foreground color, and a 4-byte overhead for the background color, with this function.

DdfSetColor - Errors

Possible returns from [WinGetLastError](#)

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

HMERR_DDF_BACKCOLOR (0x3003)
The background color is not valid.

HMERR_DDF_FORECOLOR (0x3004)
The foreground color is not valid.

DdfSetColor - Related Functions

Related Functions

- [DdfBeginList](#)
 - [DdfBitmap](#)
 - [DdfEndList](#)
 - [DdfHyperText](#)
 - [DdfInform](#)
 - [DdfInitialize](#)
 - [DdfListItem](#)
 - [DdfMetafile](#)
 - [DdfPara](#)
 - [DdfSetColor](#)
 - [DdfSetFont](#)
 - [DdfSetFontStyle](#)
 - [DdfSetFormat](#)
 - [DdfSetTextAlign](#)
 - [DdfText](#)
-

DdfSetColor - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses [DdfPara](#) to start a new paragraph, [DdfSetFont](#) and [DdfSetFontStyle](#) to have the text displayed in a large, bold Courier font, [DdfSetColor](#) to change the text color, and [DdfText](#) to place text in the buffer. For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEMGR    /* Message management */
#define INCL_DDF              /* Dynamic Data Facility */
```

```

#include <os2.h>
#include <pmhelp.h>

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND    hwndParent;
    HWND    hwndInstance;    /* help instance window */
    Hddf    hDdf;            /* DDF handle */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                          MPFROMSHORT( HMQW_INSTANCE ), NULL );

        /* Allocate 1K Buffer (default) */
        hDdf = DdfInitialize(
            hwndInstance,    /* Handle of help instance */
            0L,              /* Default buffer size */
            0L               /* Default increment */
        );

        if (hDdf == NULLHANDLE)    /* Check return code */
        {
            return (MRESULT)FALSE;
        }

        /* create paragraph in DDF buffer */
        if( !DdfPara( hDdf ) )
        {
            return (MRESULT)FALSE;
        }

        /* Change to large (100 x 100 dimensions) Courier font */
        if( !DdfSetFont( hDdf, "Courier", 100L, 100L ) )
        {
            return (MRESULT)FALSE;
        }

        /* make the font BOLDFACE */
        if( !DdfSetFontStyle( hDdf, FM_SEL_BOLD ) )
        {
            return (MRESULT)FALSE;
        }

        /* make the text display as BLUE on a PALE GRAY background */
        if( !DdfSetColor( hDdf, CLR_PALEGRAY, CLR_BLUE ) )
        {
            return (MRESULT)FALSE;
        }

        /* Write data into the buffer */
        if ( !DdfText( hDdf, "Sample Text" ) )
        {
            return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
    return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}

```

DdfSetColor - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DdfSetFont

DdfSetFont - Syntax

This function specifies a text font in the DDF buffer.

```
#define INCL_DDF
#include <os2.h>

HDDF      hddf;          /* Handle to DDF returned by DdfInitialize. */
PCSZ      pszFaceName;   /* Pointer to font name. */
ULONG     ulWidth;       /* Font width in points. */
ULONG     ulHeight;      /* Font height in points. */
BOOL      rc;            /* Success indicator. */

rc = DdfSetFont(hddf, pszFaceName, ulWidth,
               ulHeight);
```

DdfSetFont Parameter - hddf

hddf ([HDDF](#)) - input
Handle to DDF returned by [DdfInitialize](#).

DdfSetFont Parameter - pszFaceName

pszFaceName ([PCSZ](#)) - input
Pointer to font name.

This parameter can be specified in two ways:

- An ASCIIZ string specifying the font name.
 - NULL or "default" to specify the default font.
-

DdfSetFont Parameter - ulWidth

ulWidth ([ULONG](#)) - input
Font width in points.

A point is approximately 1/72 of an inch.

DdfSetFont Parameter - ulHeight

ulHeight ([ULONG](#)) - input
Font height in points.

DdfSetFont Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfSetFont - Parameters

hddf ([HDDF](#)) - input
Handle to DDF returned by [DdfInitialize](#).

pszFaceName ([PCSZ](#)) - input
Pointer to font name.

This parameter can be specified in two ways:

- An ASCIIZ string specifying the font name.
- NULL or "default" to specify the default font.

ulWidth ([ULONG](#)) - input
Font width in points.

A point is approximately 1/72 of an inch.

ulHeight ([ULONG](#)) - input
Font height in points.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfSetFont - Errors

Possible returns from [WinGetLastError](#)

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

DdfSetFont - Related Functions

Related Functions

- [DdfBeginList](#)
 - [DdfBitmap](#)
 - [DdfEndList](#)
 - [DdfHyperText](#)
 - [DdfInform](#)
 - [DdfInitialize](#)
 - [DdfListItem](#)
 - [DdfMetafile](#)
 - [DdfPara](#)
 - [DdfSetColor](#)
 - [DdfSetFont](#)
 - [DdfSetFontStyle](#)
 - [DdfSetFormat](#)
 - [DdfSetTextAlign](#)
 - [DdfText](#)
-

DdfSetFont - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses [DdfPara](#) to start a new paragraph, [DdfSetFont](#) and [DdfSetFontStyle](#) to have the text displayed in a large, bold Courier font, [DdfSetColor](#) to change the text color, and [DdfText](#) to place text in the buffer. For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEGR     /* Message management */
#define INCL_DDF              /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND    hwndParent;
    HWND    hwndInstance; /* help instance window */
    Hddf     hDdf;         /* DDF handle */

    switch( ulMsg )
    {
        case HM_QUERY_DDF_DATA:
            /* get the help instance */
            hwndParent = WinQueryWindow( hwnd, QW_PARENT );
            hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
            hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                             MPFROMSHORT( HMQW_INSTANCE ), NULL );

            /* Allocate 1K Buffer (default) */
```

```

hDdf = DdfInitialize(
    hwndInstance, /* Handle of help instance */
    0L,           /* Default buffer size */
    0L,           /* Default increment */
);

if (hDdf == NULLHANDLE) /* Check return code */
{
    return (MRESULT)FALSE;
}

/* create paragraph in DDF buffer */
if( !DdfPara( hDdf ) )
{
    return (MRESULT)FALSE;
}

/* Change to large (100 x 100 dimensions) Courier font */
if( !DdfSetFont( hDdf, "Courier", 100L, 100L ) )
{
    return (MRESULT)FALSE;
}

/* make the font BOLDFACE */
if( !DdfSetFontStyle( hDdf, FM_SEL_BOLD ) )
{
    return (MRESULT)FALSE;
}

/* make the text display as BLUE on a PALE GRAY background */
if( !DdfSetColor( hDdf, CLR_PALEGRAY, CLR_BLUE ) )
{
    return (MRESULT)FALSE;
}

/* Write data into the buffer */
if ( !DdfText(hDdf, "Sample Text") )
{
    return (MRESULT)FALSE;
}

return (MRESULT)hDdf;
}
return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}

```

DdfSetFont - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DdfSetFontStyle

DdfSetFontStyle - Syntax

This function specifies a text font style in the DDF buffer.

```
#define INCL_DDF
#include <os2.h>

HDDF    hddf;          /* Handle to DDF returned by DdfInitialize. */
ULONG    fFontStyle;    /* Font style flag. */
BOOL     rc;            /* Success indicator. */

rc = DdfSetFontStyle(hddf, fFontStyle);
```

DdfSetFontStyle Parameter - hddf

hddf (**HDDF**) - input
Handle to DDF returned by [DdfInitialize](#).

DdfSetFontStyle Parameter - fFontStyle

fFontStyle (**ULONG**) - input
Font style flag.

A NULL value for this parameter will set the font-style back to the default. Any of the following values can be specified:

FM_SEL_ITALIC
FM_SEL_BOLD
FM_SEL_UNDERSCORE

These values can be "ORed" together to combine different font styles.

Note: There is a 4-byte ESC code overhead in the DDF internal buffer for *fFontStyle*.

DdfSetFontStyle Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE
Successful completion.
FALSE
Error occurred.

DdfSetFontStyle - Parameters

hddf ([HDDF](#)) - input
Handle to DDF returned by [DdfInitialize](#).

fFontStyle ([ULONG](#)) - input
Font style flag.

A NULL value for this parameter will set the font-style back to the default. Any of the following values can be specified:

FM_SEL_ITALIC
FM_SEL_BOLD
FM_SEL_UNDERSCORE

These values can be "ORed" together to combine different font styles.

Note: There is a 4-byte ESC code overhead in the DDF internal buffer for *fFontStyle*.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DdfSetFontStyle - Errors

Possible returns from [WinGetLastError](#)

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

HMERR_DDF_FONTSTYLE (0x3005)
The font style is not valid.

DdfSetFontStyle - Related Functions

Related Functions

- [DdfBeginList](#)
- [DdfBitmap](#)
- [DdfEndList](#)
- [DdfHyperText](#)
- [DdfInform](#)
- [DdfInitialize](#)
- [DdfListItem](#)
- [DdfMetafile](#)
- [DdfPara](#)
- [DdfSetColor](#)
- [DdfSetFont](#)
- [DdfSetFontStyle](#)
- [DdfSetFormat](#)
- [DdfSetTextAlign](#)
- [DdfText](#)

DdfSetFontStyle - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses DdfPara to start a new paragraph, DdfSetFont and DdfSetFontStyle to have the text displayed in a large, bold Courier font, DdfSetColor to change the text color, and DdfText to place text in the buffer. For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEMGR    /* Message management */
#define INCL_DDF              /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND    hwndParent;
    HWND    hwndInstance;    /* help instance window */
    Hddf     hDdf;           /* DDF handle */

    switch( ulMsg )
    {
        case HM_QUERY_DDF_DATA:
            /* get the help instance */
            hwndParent = WinQueryWindow( hwnd, QW_PARENT );
            hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
            hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                             MPFROMSHORT( HMQW_INSTANCE ), NULL );

            /* Allocate 1K Buffer (default) */
            hDdf = DdfInitialize(
                hwndInstance,    /* Handle of help instance */
                0L,             /* Default buffer size */
                0L,             /* Default increment */
            );

            if (hDdf == NULLHANDLE)    /* Check return code */
            {
                return (MRESULT)FALSE;
            }

            /* create paragraph in DDF buffer */
            if( !DdfPara( hDdf ) )
            {
                return (MRESULT)FALSE;
            }

            /* Change to large (100 x 100 dimensions) Courier font */
            if( !DdfSetFont( hDdf, "Courier", 100L, 100L ) )
            {
                return (MRESULT)FALSE;
            }

            /* make the font BOLDFACE */
            if( !DdfSetFontStyle( hDdf, FM_SEL_BOLD ) )
            {
                return (MRESULT)FALSE;
            }

            /* make the text display as BLUE on a PALE GRAY background */
            if( !DdfSetColor( hDdf, CLR_PALEGRAY, CLR_BLUE ) )
            {
                return (MRESULT)FALSE;
            }

            /* Write data into the buffer */
            if ( !DdfText( hDdf, "Sample Text" ) )
            {
                return (MRESULT)FALSE;
            }

            return (MRESULT)hDdf;
        }
    }
    return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}
```

```
}
```

DdfSetFontStyle - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

DdfSetFormat

DdfSetFormat - Syntax

This function is used to turn formatting off or on. It corresponds to the **:lines.** tag.

```
#define INCL_DDF
#include <os2.h>

HDDF    hddf;          /* Handle to DDF returned by DdfInitialize. */
ULONG   fFormatType;   /* Formatting-activation flag. */
BOOL    rc;            /* Success indicator. */

rc = DdfSetFormat(hddf, fFormatType);
```

DdfSetFormat Parameter - hddf

hddf ([HDDF](#)) - input

Handle to DDF returned by [DdfInitialize](#).

DdfSetFormat Parameter - fFormatType

fFormatType ([ULONG](#)) - input

Formatting-activation flag.

Only the following constants may be used in this parameter:

- | | |
|-------|----------------------|
| TRUE | Turn formatting on. |
| FALSE | Turn formatting off. |

DdfSetFormat Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

- | | |
|-------|-----------------------|
| TRUE | Successful completion |
| FALSE | Error occurred. |

DdfSetFormat - Parameters

hddf ([HDDF](#)) - input
Handle to DDF returned by [DdfInitialize](#).

fFormatType ([ULONG](#)) - input
Formatting-activation flag.

Only the following constants may be used in this parameter:

- | | |
|-------|----------------------|
| TRUE | Turn formatting on. |
| FALSE | Turn formatting off. |

rc ([BOOL](#)) - returns
Success indicator.

- | | |
|-------|-----------------------|
| TRUE | Successful completion |
| FALSE | Error occurred. |

DdfSetFormat - Remarks

If formatting is ON, there is a 3-byte ESC code overhead in the DDF internal buffer for this function. Otherwise, there is a 4-byte ESC code overhead.

DdfSetFormat - Errors

Possible returns from [WinGetLastError](#)

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

DdfSetFormat - Related Functions

Related Functions

- [DdfBeginList](#)
- [DdfBitmap](#)
- [DdfEndList](#)
- [DdfHyperText](#)
- [DdfInform](#)
- [DdfInitialize](#)
- [DdfListItem](#)
- [DdfMetafile](#)
- [DdfPara](#)
- [DdfSetColor](#)
- [DdfSetFont](#)
- [DdfSetFontStyle](#)
- [DdfSetFormat](#)
- [DdfSetTextAlign](#)
- [DdfText](#)

DdfSetFormat - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses [DdfSetTextAlign](#) to specify left justified text in the DDF buffer when formatting is OFF. The example then uses [DdfSetFormat](#) to turn off formatting for text in the DDF buffer (corresponds to the IPF lines tag). For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEMGR    /* Message management */
#define INCL_GDIPRIMITIVES    /* Drawing Primitives/Attributes */
#define INCL_DDF               /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND    hwndParent;
    HWND    hwndInstance; /* help instance window */
    Hddf    hDdf; /* DDF handle */

    switch( ulMsg )
    {
        case HM_QUERY_DDF_DATA:
            /* get the help instance */
            hwndParent = WinQueryWindow( hwnd, QW_PARENT );
            hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
            hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                             MPFROMSHORT( HMQW_INSTANCE ), NULL );

            /* Allocate 1K Buffer (default) */
            hDdf = DdfInitialize(
                hwndInstance, /* Handle of help instance */
                0L,           /* Default buffer size */
                0L,           /* Default increment */
            );
    }
}
```

```

        if (hDdf == NULLHANDLE)          /* Check return code          */
        {
            return (MRESULT)FALSE;
        }

        /* left justify text when formatting is OFF */
        if (!DdfSetTextAlign(hDdf, TA_LEFT))
        {
            return (MRESULT)FALSE;
        }

        /* turn formatting OFF */
        if (!DdfSetFormat(hDdf, FALSE))
        {
            return (MRESULT)FALSE;
        }

        if (!DdfText(hDdf,
            "Format OFF: This text should be Left Aligned!\n"))
        {
            return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
    return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}

```

DdfSetFormat - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DdfSetTextAlign

DdfSetTextAlign - Syntax

This function defines whether left, center, or right text justification is to be used when text formatting is off.

```

#define INCL_DDF
#include <os2.h>

HDDF      hddf;      /* Handle to DDF returned by DdfInitialize. */
ULONG     fAlign;    /* Justification flag. */
BOOL      rc;        /* Success indicator. */

rc = DdfSetTextAlign(hddf, fAlign);

```

DdfSetTextAlign Parameter - hddf

hddf ([HDDF](#)) - input
Handle to DDF returned by [DdfInitialize](#).

DdfSetTextAlign Parameter - fAlign

fAlign ([ULONG](#)) - input
Justification flag.

Only the following constants may be used:

TA_LEFT	Left-justify text.
TA_RIGHT	Right-justify text.
TA_CENTER	Center text.

DdfSetTextAlign Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DdfSetTextAlign - Parameters

hddf ([HDDF](#)) - input
Handle to DDF returned by [DdfInitialize](#).

fAlign ([ULONG](#)) - input
Justification flag.

Only the following constants may be used:

TA_LEFT	Left-justify text.
TA_RIGHT	

TA_CENTER	Right-justify text.
	Center text.
rc (BOOL) - returns	
	Success indicator.
TRUE	Successful completion.
FALSE	Error occurred.

DdfSetTextAlign - Remarks

It should be called before DdfSetFormat is called to turn off text formatting, and should not be called again until formatting is turned back on. Note that leading and trailing spaces are not stripped from the text as a result of this alignment.

DdfSetTextAlign - Errors

Possible returns from [WinGetLastError](#)

HMERR_DDF_ALIGN_TYPE (0x3002)
The alignment type is not valid.

DdfSetTextAlign - Related Functions

Related Functions

- [DdfBeginList](#)
- [DdfBitmap](#)
- [DdfEndList](#)
- [DdfHyperText](#)
- [DdfInform](#)
- [DdfInitialize](#)
- [DdfListItem](#)
- [DdfMetafile](#)
- [DdfPara](#)
- [DdfSetColor](#)
- [DdfSetFont](#)
- [DdfSetFontStyle](#)
- [DdfSetFormat](#)
- [DdfSetTextAlign](#)
- [DdfText](#)

DdfSetTextAlign - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses DdfSetTextAlign to specify left justified text in the DDF buffer when formatting is OFF. The example then uses [DdfSetFormat](#) to turn off formatting for text in the DDF buffer (corresponds to the IPF lines tag). For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```

#define INCL_WINWINDOWMGR          /* General window management */
#define INCL_WINMESSAGEMGR        /* Message management */
#define INCL_GPIPRIMITIVES        /* Drawing Primitives/Attributes*/
#define INCL_DDF                  /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND    hwndParent;
    HWND    hwndInstance;    /* help instance window */
    Hddf    hDdf;            /* DDF handle */

    switch( ulMsg )
    {
        case HM_QUERY_DDF_DATA:
            /* get the help instance */
            hwndParent = WinQueryWindow( hwnd, QW_PARENT );
            hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
            hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                MPFROMSHORT( HMQW_INSTANCE ), NULL );

            /* Allocate 1K Buffer (default) */
            hDdf = DdfInitialize(
                hwndInstance, /* Handle of help instance */
                0L,          /* Default buffer size */
                0L,          /* Default increment */
            );

            if (hDdf == NULLHANDLE) /* Check return code */
            {
                return (MRESULT)FALSE;
            }

            /* left justify text when formatting is OFF */
            if (!DdfSetTextAlign(hDdf, TA_LEFT))
            {
                return (MRESULT)FALSE;
            }

            /* turn formatting OFF */
            if (!DdfSetFormat(hDdf, FALSE))
            {
                return (MRESULT)FALSE;
            }

            if (!DdfText(hDdf,
                "Format OFF: This text should be Left Aligned!\n"))
            {
                return (MRESULT)FALSE;
            }

            return (MRESULT)hDdf;
        }
    }
    return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}

```

DdfSetTextAlign - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DdfText

DdfText - Syntax

This function adds text to the DDF buffer.

```
#define INCL_DDF
#include <os2.h>

HDDF    hddf;    /* Handle to DDF returned by DdfInitialize. */
PCSZ    pszText; /* Pointer to the text buffer to be formatted. */
BOOL    rc;      /* Success indicator. */

rc = DdfText(hddf, pszText);
```

DdfText Parameter - hddf

hddf (**HDDF**) - input
Handle to DDF returned by [DdfInitialize](#).

DdfText Parameter - pszText

pszText (**PCSZ**) - input
Pointer to the text buffer to be formatted.

Note: There is a 3-byte ESC code overhead in the DDF internal buffer for each word in the text buffer. There is a 1-byte ESC code overhead for each blank and for each newline character.

DdfText Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion.
FALSE	

Error occurred.

DdfText - Parameters

hddf ([HDDF](#)) - input
Handle to DDF returned by [DdfInitialize](#).

pszText ([PCSZ](#)) - input
Pointer to the text buffer to be formatted.

Note: There is a 3-byte ESC code overhead in the DDF internal buffer for each word in the text buffer. There is a 1-byte ESC code overhead for each blank and for each newline character.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DdfText - Related Functions

Related Functions

- [DdfBeginList](#)
 - [DdfBitmap](#)
 - [DdfEndList](#)
 - [DdfHyperText](#)
 - [DdfInform](#)
 - [DdfInitialize](#)
 - [DdfListItem](#)
 - [DdfMetafile](#)
 - [DdfPara](#)
 - [DdfSetColor](#)
 - [DdfSetFont](#)
 - [DdfSetFontStyle](#)
 - [DdfSetFormat](#)
 - [DdfSetTextAlign](#)
 - [DdfText](#)
-

DdfText - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses [DdfPara](#) to start a new paragraph, [DdfSetFont](#) and [DdfSetFontStyle](#) to have the text displayed in a large, bold Courier font, [DdfSetColor](#) to change the text color, and [DdfText](#) to place text in the buffer. For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEMGR    /* Message management */
#define INCL_DDF               /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>
```

```

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND    hwndParent;
    HWND    hwndInstance;    /* help instance window */
    Hddf    hDdf;            /* DDF handle */

    switch( ulMsg )
    {
        case HM_QUERY_DDF_DATA:
            /* get the help instance */
            hwndParent = WinQueryWindow( hwnd, QW_PARENT );
            hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
            hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                             MPFROMSHORT( HMQW_INSTANCE ), NULL );

            /* Allocate 1K Buffer (default) */
            hDdf = DdfInitialize(
                hwndInstance, /* Handle of help instance */
                0L,          /* Default buffer size */
                0L,          /* Default increment */
                );

            if (hDdf == NULLHANDLE) /* Check return code */
            {
                return (MRESULT)FALSE;
            }

            /* create paragraph in DDF buffer */
            if( !DdfPara( hDdf ) )
            {
                return (MRESULT)FALSE;
            }

            /* Change to large (100 x 100 dimensions) Courier font */
            if( !DdfSetFont( hDdf, "Courier", 100L, 100L ) )
            {
                return (MRESULT)FALSE;
            }

            /* make the font BOLDFACE */
            if( !DdfSetFontStyle( hDdf, FM_SEL_BOLD ) )
            {
                return (MRESULT)FALSE;
            }

            /* make the text display as BLUE on a PALE GRAY background */
            if( !DdfSetColor( hDdf, CLR_PALEGRAY, CLR_BLUE ) )
            {
                return (MRESULT)FALSE;
            }

            /* Write data into the buffer */
            if ( !DdfText(hDdf, "Sample Text") )
            {
                return (MRESULT)FALSE;
            }

            return (MRESULT)hDdf;
        }
    }
    return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}

```

DdfText - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Example Code](#)

Hooks and Procedures

This section describes dialog procedures, window procedures, and hooks. It shows the input parameters and returns that the operating system expects an application to use in application procedures and that can be called by the operating system in response to certain events.

Procedures and hooks are application code that is called by the system in response to certain events.

The names and parameter lists of functions are contained in header files that are incorporated into the application when it is compiled. Their addresses are contained in .LIB files that are incorporated at link time.

The names of procedures and hooks are defined by the application, and their parameter lists are defined by the system. Function prototypes for these procedures and hooks are in PMWIN.H. The prototypes have sample names that can be changed by the programmer before they are inserted into the application source code.

The application passes the address of these procedures and hooks in the following ways:

Dialog procedures	During the WinLoadDlg, WinDlgBox, WinFileDlg, or WinFontDlg function
Window procedures	During the WinRegisterClass or WinSubclassWindow functions
Hooks	During the WinSetHook function
Thunks	During the WinSetClassThunkProc or WinSetWindowThunkProc functions.

Hook Functions

This section of the chapter describes the hook functions that the application can provide:

- [CheckMsgFilterHook](#)
 - [CodePageChangedHook](#)
 - [DestroyWindowHook](#)
 - [DialogProc](#)
 - [FindWordHook](#)
 - [FlushBufHook](#)
 - [HelpHook](#)
 - [InputHook](#)
 - [JournalPlaybackHook](#)
 - [JournalRecordHook](#)
 - [LoaderHook](#)
 - [LockupHook](#)
 - [MsgControlHook](#)
 - [MsgFilterHook](#)
 - [MsgInputHook](#)
 - [ProgramListEntryHook](#)
 - [ProgramListExitHook](#)
 - [RegisterUserHook](#)
 - [SendMsgHook](#)
 - [ThunkProc](#)
 - [WindowDCHook](#)
 - [WndProc](#)
-

CheckMsgFilterHook

CheckMsgFilterHook - Syntax

This hook is called whenever [WinGetMsg](#), [WinWaitMsg](#), or [WinPeekMsg](#) are used to filter message identities.

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
PQMSG    pQmsg;        /* Pointer to the QMSG structure of the message currently being reviewed. */
ULONG    usFirst;      /* First message identity specified on a call to the WinGetMsg, WinPeekMsg or WinWaitMsg f
ULONG    usLast;       /* Last message identity specified on a call to the WinGetMsg, WinPeekMsg or WinWaitMsg fun
ULONG    fOptions;     /* Message removal options. */
BOOL     rc;           /* Processing indicator. */

rc = CheckMsgFilterHook(hab, pQmsg, usFirst,
                        usLast, fOptions);
```

CheckMsgFilterHook Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

CheckMsgFilterHook Parameter - pQmsg

pQmsg ([PQMSG](#)) - input
Pointer to the [QMSG](#) structure of the message currently being reviewed.

CheckMsgFilterHook Parameter - usFirst

usFirst ([ULONG](#)) - input
First message identity specified on a call to the [WinGetMsg](#), [WinPeekMsg](#) or [WinWaitMsg](#) function.

CheckMsgFilterHook Parameter - usLast

usLast ([ULONG](#)) - input
Last message identity specified on a call to the [WinGetMsg](#), [WinPeekMsg](#) or [WinWaitMsg](#) function.

CheckMsgFilterHook Parameter - fOptions

fOptions ([ULONG](#)) - input
Message removal options.

PM_REMOVE Message is being removed from queue
PM_NOREMOVE Message is not being removed from queue.

CheckMsgFilterHook Return Value - rc

rc ([BOOL](#)) - returns
Processing indicator.

TRUE
The message is accepted by the filtering. Any further Check Message Filter Hooks in the chain are ignored, any filtering specified by the *uIfirst* and *uIfirst* parameters of the [WinGetMsg](#), [WinPeekMsg](#) or [WinWaitMsg](#) functions are ignored, and processing of the message continues.

A hook that always returns TRUE effectively switches off message filtering.

FALSE
The message is passed on to the next Check Message Filter Hook in the chain. If the end of the chain has been reached, the filtering specified by the *uIfirst* and *uIfirst* parameters of the [WinGetMsg](#), [WinPeekMsg](#) or [WinWaitMsg](#) functions is applied.

CheckMsgFilterHook - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

pQmsg ([PQMSG](#)) - input
Pointer to the [QMSG](#) structure of the message currently being reviewed.

usFirst ([ULONG](#)) - input
First message identity specified on a call to the [WinGetMsg](#), [WinPeekMsg](#) or [WinWaitMsg](#) function.

usLast ([ULONG](#)) - input
Last message identity specified on a call to the [WinGetMsg](#), [WinPeekMsg](#) or [WinWaitMsg](#) function.

fOptions ([ULONG](#)) - input
Message removal options.

PM_REMOVE Message is being removed from queue
PM_NOREMOVE Message is not being removed from queue.

rc ([BOOL](#)) - returns
Processing indicator.

TRUE

The message is accepted by the filtering. Any further Check Message Filter Hooks in the chain are ignored, any filtering specified by the *uIfirst* and *uIfast* parameters of the [WinGetMsg](#), [WinPeekMsg](#) or [WinWaitMsg](#) functions are ignored, and processing of the message continues.

A hook that always returns TRUE effectively switches off message filtering.

FALSE

The message is passed on to the next Check Message Filter Hook in the chain. If the end of the chain has been reached, the filtering specified by the *uIfirst* and *uIfast* parameters of the [WinGetMsg](#), [WinPeekMsg](#) or [WinWaitMsg](#) functions is applied.

CheckMsgFilterHook - Remarks

This hook enables an application to apply a very specific message filtering, for example, based on the values of message parameters.

This hook is called after window handle filtering and before message filtering. Window handle filtering is controlled by the *hwndFilter* parameter of the [WinGetMsg](#) or [WinPeekMsg](#) functions. Message filtering is controlled by the *uIfirst* and *uIfast* parameters of the [WinGetMsg](#), [WinPeekMsg](#) or [WinWaitMsg](#) functions.

This hook is called if the message passes window handle filtering and if non-null message filtering is specified. This means that, on entry to this hook:

- The *hwndFilter* parameter of the [WinGetMsg](#) or [WinPeekMsg](#) function is either NULLHANDLE or it specifies the window (or a parent of the window) referenced in the *pQmsg* structure.
- At least one of the *usFirst* and *usLast* parameters are nonzero.
- The field of the *pQmsg* structure might or might not lie inside the range specified by the *usFirst* and *usLast* parameters.

CheckMsgFilterHook - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

CodePageChangedHook

CodePageChangedHook - Syntax

This hook notifies that a message queue code page has been changed.

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HMQ      hmq;          /* Message-queue handle. */
ULONG    usOldCodePage; /* Previous code page. */
ULONG    usNewCodePage; /* New code page. */

CodePageChangedHook(hmq, usOldCodePage, usNewCodePage);
```

CodePageChangedHook Parameter - hmq

hmq ([HMQ](#)) - input
Message-queue handle.

The handle of the message queue that is changing its code page.

CodePageChangedHook Parameter - usOldCodePage

usOldCodePage ([ULONG](#)) - input
Previous code page.

CodePageChangedHook Parameter - usNewCodePage

usNewCodePage ([ULONG](#)) - input
New code page.

CodePageChangedHook - Return Value

There is no return value for this hook.

CodePageChangedHook - Parameters

hmq ([HMQ](#)) - input

Message-queue handle.

The handle of the message queue that is changing its code page.

usOldCodePage ([ULONG](#)) - input
Previous code page.

usNewCodePage ([ULONG](#)) - input
New code page.

There is no return value for this hook.

CodePageChangedHook - Remarks

This hook is sent to all hooks chained under HK_CODEPAGECHANGED, regardless of the return value.

The new code page is set before this hook is called.

CodePageChangedHook - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

DestroyWindowHook

DestroyWindowHook - Syntax

This hook is called whenever a window is destroyed.

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
HWND     hwnd;         /* The handle of the window being destroyed. */
ULONG    ulReserved;   /* Reserved. */
BOOL     rc;           /* Success indicator. */

rc = DestroyWindowHook(hab, hwnd, ulReserved);
```

DestroyWindowHook Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

DestroyWindowHook Parameter - hwnd

hwnd ([HWND](#)) - input
The handle of the window being destroyed.

DestroyWindowHook Parameter - ulReserved

ulReserved ([ULONG](#)) - input
Reserved.

DestroyWindowHook Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DestroyWindowHook - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

hwnd ([HWND](#)) - input
The handle of the window being destroyed.

ulReserved ([ULONG](#)) - input
Reserved.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DestroyWindowHook - Remarks

This hook is sent after the [WM_DESTROY](#) message has been sent and just before the window becomes invalid.

DestroyWindowHook - Related Messages

Related Messages

- [WM_DESTROY](#)
-

DestroyWindowHook - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Messages](#)
[Glossary](#)

DialogProc

DialogProc - Syntax

This is a window procedure that automatically subclasses each instance of a dialog box.

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND      hwnd;          /* Handle of the window to which the message applies. */
USHORT    usmsg;          /* Message identity. */
MPARAM    mpParam1;       /* Message parameter 1. */
MPARAM    mpParam2;       /* Message parameter 2. */
MRESULT    mresReply;     /* Message-return data. */
```



```
mresReply = DialogProc(hwnd, usmsg, mpParam1,  
                        mpParam2);
```

DialogProc Parameter - hwnd

hwnd (**HWND**) - input
Handle of the window to which the message applies.

DialogProc Parameter - usmsg

usmsg (**USHORT**) - input
Message identity.

DialogProc Parameter - mpParam1

mpParam1 (**MPARAM**) - input
Message parameter 1.

DialogProc Parameter - mpParam2

mpParam2 (**MPARAM**) - input
Message parameter 2.

DialogProc Return Value - mresReply

mresReply (**MRESULT**) - returns
Message-return data.

DialogProc - Parameters

hwnd ([HWND](#)) - input
Handle of the window to which the message applies.

usmsg ([USHORT](#)) - input
Message identity.

mpParam1 ([MPARAM](#)) - input
Message parameter 1.

mpParam2 ([MPARAM](#)) - input
Message parameter 2.

mresReply ([MRESULT](#)) - returns
Message-return data.

DialogProc - Remarks

This procedure is the same as any other window procedure, except that it can receive predefined window messages specific to dialog box windows.

Note: It does *not* receive the [WM_CREATE](#) message, but the same information is carried by the [WM_INITDLG](#) message, that is generated during the creation of a dialog-box window.

hwnd is always the window handle of the dialog-box window.

The dialog procedure typically processes only some of the messages passed to it. Any messages that it does not process must be passed to [WinDefFileDlgProc](#) if the dialog box is the standard file selection dialog, [WinDefFontDlgProc](#) if the dialog box is the standard font selection dialog box, or for all other dialog boxes, [WinDefDlgProc](#) (not [WinDefWindowProc](#)), because these perform the standard dialog-box processing for those messages.

DialogProc - Related Messages

Related Messages

- [WM_CREATE](#)
- [WM_INITDLG](#)

DialogProc - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Messages](#)
[Glossary](#)

FindWordHook

FindWordHook - Syntax

This hook allows an application to control where the [WinDrawText](#) function breaks a character string that is too long for the drawing rectangle.

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

USHORT    usCodepage; /* Codepage to use. */
PSZ       pszText;    /* Text to break. */
ULONG     cb;         /* Maximum text size. */
ULONG     ich;        /* Break near here. */
PULONG    pichStart;  /* Where break began. */
PULONG    pichEnd;    /* Where break ended. */
PULONG    pichNext;   /* Where next word begins. */
BOOL      rc;         /* Success indicator. */

rc = FindWordHook(usCodepage, pszText, cb,
                 ich, pichStart, pichEnd, pichNext);
```

FindWordHook Parameter - usCodepage

usCodepage ([USHORT](#)) - input
Codepage to use.

This parameter contains the codepage identifier of the string to be formatted.

FindWordHook Parameter - pszText

pszText ([PSZ](#)) - input
Text to break.

This parameter contains a pointer to the actual string.

FindWordHook Parameter - cb

cb ([ULONG](#)) - input
Maximum text size.

This parameter contains a value specifying the number of bytes in the string.

FindWordHook Parameter - ich

ich ([ULONG](#)) - input
Break near here.

This parameter contains the index of the character in the string that intersects the right edge of the drawing rectangle.

FindWordHook Parameter - pichStart

pichStart ([PULONG](#)) - output
Where break began.

This parameter contains the index of the starting character of the intersecting word.

FindWordHook Parameter - pichEnd

pichEnd ([PULONG](#)) - output
Where break ended.

This parameter contains the index of the ending character of the intersecting word.

FindWordHook Parameter - pichNext

pichNext ([PULONG](#)) - output
Where next word begins.

This parameter contains the index of the starting character of the next word in the string.

FindWordHook Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE

If the find-word hook function returns TRUE, [WinDrawText](#) will only draw the string up to, but not including, the specified word.

FALSE

If the find-word hook function returns FALSE, [WinDrawText](#) formats the string in the default manner.

FindWordHook - Parameters

usCodepage ([USHORT](#)) - input
Codepage to use.

This parameter contains the codepage identifier of the string to be formatted.

pszText ([PSZ](#)) - input
Text to break.

This parameter contains a pointer to the actual string.

cb ([ULONG](#)) - input
Maximum text size.

This parameter contains a value specifying the number of bytes in the string.

ich ([ULONG](#)) - input
Break near here.

This parameter contains the index of the character in the string that intersects the right edge of the drawing rectangle.

pichStart ([PULONG](#)) - output
Where break began.

This parameter contains the index of the starting character of the intersecting word.

pichEnd ([PULONG](#)) - output
Where break ended.

This parameter contains the index of the ending character of the intersecting word.

pichNext ([PULONG](#)) - output
Where next word begins.

This parameter contains the index of the starting character of the next word in the string.

rc ([BOOL](#)) - returns
Success indicator.

TRUE

If the find-word hook function returns TRUE, [WinDrawText](#) will only draw the string up to, but not including, the specified word.

FALSE

If the find-word hook function returns FALSE, [WinDrawText](#) formats the string in the default manner.

FindWordHook - Remarks

The system calls this hook from within the [WinDrawText](#) function, if the DT_WORDBREAK flag is set. It lets the application have control of where the function [WinDrawText](#) should break for a string that is too long.

FindWordHook - Topics

Select an item:

- Syntax
- Parameters
- Returns
- Remarks
- Glossary

FlushBufHook

FlushBufHook - Syntax

This function is specific to OS/2 Version 2.1 or higher.

This hook allows applications to save data before the system reboots.

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB hab; /* Application anchor block. */
BOOL rc; /* Return code. */

rc = FlushBufHook(hab);
```

FlushBufHook Parameter - hab

hab (HAB) - input
Application anchor block.

FlushBufHook Return Value - rc

rc (BOOL) - returns
Return code.

TRUE	Successful completion.
FALSE	An error occurred.

FlushBufHook - Parameters

hab ([HAB](#)) - input
Application anchor block.

rc ([BOOL](#)) - returns
Return code.

TRUE	Successful completion.
FALSE	An error occurred.

FlushBufHook - Remarks

This hook is called to notify applications that the system is rebooting due to a Ctrl+Alt+Del sequence being entered. It enables applications to write existing information to the hardfile immediately, avoiding loss of data before the system reboots.

Note: Do not use any of the Win or Gpi functions inside the hook routine. At the point in time that the hook routine is executing, these sub-systems may not be fully available because they might be partially shutdown.

FlushBufHook - Related Functions

- Related Functions**
- [WinSetHook](#)

FlushBufHook - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Related Functions](#)
 - [Glossary](#)

HelpHook

HelpHook - Syntax

This hook processes help requests.

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
ULONG     usMode;       /* Help mode. */
ULONG     idTopic;      /* Topic identifier. */
ULONG     idSubTopic;   /* Subtopic identifier. */
PRECTL    prcPosition; /* Rectangle. */
BOOL      rc;           /* Indicator as to whether next hook in the chain is called. */

rc = HelpHook(hab, usMode, idTopic, idSubTopic,
               prcPosition);
```

HelpHook Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

HelpHook Parameter - usMode

usMode (**ULONG**) - input
Help mode.

This has one of the following values, indicating the mode from which help has been requested:

HLPM_FRAME	Standard (standard window)
HLPM_WINDOW	Standard (standard window)
HLPM_MENU	Menu mode

HelpHook Parameter - idTopic

idTopic (**ULONG**) - input
Topic identifier.

- In menu mode this is a pull-down window identity
- In message-box mode this is the message-box identity
- In standard mode this is a window identity.

HelpHook Parameter - idSubTopic

idSubTopic ([ULONG](#)) - input
Subtopic identifier.

- In menu mode this is a command identity
- In message-box mode this is a control identity
- In standard mode this is the identity of the window with the focus (-1 if none).

HelpHook Parameter - prcPosition

prcPosition ([PRECTL](#)) - input
Rectangle.

This indicates the screen area (in screen coordinates) from where the help was requested. It is provided to enable the help library to avoid covering that area.

- In menu mode it is the bounding rectangle of the selected item, or of the top level menu if value of the *idSubTopic* parameter is -1.
- In message-box mode it is the bounding rectangle of the button.
- In standard mode it is the bounding rectangle of the window with the focus, or of the window sent the message if the value of the *idSubTopic* parameter is -1.

Note: The data type [WRECT](#) can also be used, if supported by the language.

HelpHook Return Value - rc

rc ([BOOL](#)) - returns
Indicator as to whether next hook in the chain is called.

The message is always passed to the application.

TRUE	The next hook in the chain is not called.
FALSE	The next hook in the chain is called.

HelpHook - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

usMode ([ULONG](#)) - input
Help mode.

This has one of the following values, indicating the mode from which help has been requested:

HLPM_FRAME	Standard (standard window)
HLPM_WINDOW	Standard (standard window)
HLPM_MENU	Menu mode

idTopic ([ULONG](#)) - input
Topic identifier.

- In menu mode this is a pull-down window identity
- In message-box mode this is the message-box identity
- In standard mode this is a window identity.

idSubTopic ([ULONG](#)) - input
Subtopic identifier.

- In menu mode this is a command identity
- In message-box mode this is a control identity
- In standard mode this is the identity of the window with the focus (-1 if none).

prcPosition ([PRECTL](#)) - input
Rectangle.

This indicates the screen area (in screen coordinates) from where the help was requested. It is provided to enable the help library to avoid covering that area.

- In menu mode it is the bounding rectangle of the selected item, or of the top level menu if value of the *idSubTopic* parameter is -1.
- In message-box mode it is the bounding rectangle of the button.
- In standard mode it is the bounding rectangle of the window with the focus, or of the window sent the message if the value of the *idSubTopic* parameter is -1.

Note: The data type [WRECT](#) can also be used, if supported by the language.

rc ([BOOL](#)) - returns
Indicator as to whether next hook in the chain is called.

The message is always passed to the application.

TRUE	The next hook in the chain is not called.
FALSE	The next hook in the chain is called.

HelpHook - Remarks

Help-processing is done in two stages. The first stage is the creation of the [WM_HELP](#) message. This is done:

- From a [WM_CHAR](#) message by ACCELERATOR table translation, when the HELP accelerator option is specified.
- From an action-bar selection, when the MIS_HELP style is specified on the action-bar button.
- From a dialog-box push button, when the BS_HELP style is specified on the push button.
- From a message box, when the MB_HELP style is specified on the message box.

The [WM_HELP](#) message is sent to the active window, but will be seen by a modal loop if one is active.

The second stage of processing of help is the processing of the [WM_HELP](#) message.

The frame window procedure sees the [WM_HELP](#) message because the frame is usually the active window. It processes the [WM_HELP](#) message as follows:

- If the window with the focus is the FID_CLIENT frame control, it passes [WM_HELP](#) to the FID_CLIENT window.
- If the parent of the window with the focus is the FID_CLIENT frame control, it calls the help hook, specifying:

```
usMode = HLPM_WINDOW  
idTopic = frame-window id  
idSubTopic = focus-window id.
```

- If the parent of the focus window is not the FID_CLIENT frame control (for example, it may be the frame itself, or a second-level dialog control), it calls the hook, specifying:

```
usMode = HLPM_WINDOW  
idTopic = focus-window parent id  
idSubTopic = focus-window id.
```

The message box window procedure sees the [WM_HELP](#) message, because it subclasses the frame window. It processes the [WM_HELP](#) message by calling the help hook, specifying:

```
usMode = HLPM_MESSAGE  
idTopic = message id  
idSubTopic = control id.
```

The menu window procedure sees the [WM_HELP](#) message because it runs a modal loop. It processes the [WM_HELP](#) message by calling the help hook, specifying:

```
usMode = HLPM_MENU  
idTopic = menu id of pulldown  
idSubTopic = menu id of item.
```

The [WinDefWindowProc](#) function sees the [WM_HELP](#) message for a FID_CLIENT window if the client does not handle it itself. It calls the help hook, specifying:

```
usMode = HLPM_WINDOW  
idTopic = active-window id  
idSubTopic = focus-window id.
```

An application sees the [WM_HELP](#) message in its dialog procedure. The application can ignore the [WM_HELP](#) message, in which case the frame-window procedure action occurs (as described above) or it can simulate a call to the help hook itself, using:

```
usMode = HLPM_APPLICATION  
idTopic = any value  
idSubTopic = any value.
```

The input focus is never given to any of the standard frame controls, so help for these cannot be obtained.

HelpHook - Related Messages

Related Messages

- [WM_CHAR](#)
- [WM_HELP](#)

HelpHook - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)

[Remarks](#)
[Related Messages](#)
[Glossary](#)

InputHook

InputHook - Syntax

This hook filters messages from the input queue.

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;      /* Anchor-block handle. */
PQMSG    pQmsg;    /* A PQMSG data structure. */
ULONG    fs;       /* Message removal options. */
BOOL     rc;       /* Processed indicator. */

rc = InputHook(hab, pQmsg, fs);
```

InputHook Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

InputHook Parameter - pQmsg

pQmsg ([PQMSG](#)) - input
A PQMSG data structure.

InputHook Parameter - fs

fs ([ULONG](#)) - input
Message removal options.

PM_REMOVE

PM_NOREMOVE Message is being removed from queue
Message is not being removed from queue.

InputHook Return Value - rc

rc (**BOOL**) - returns
Processed indicator.

TRUE The message is not passed on to the next hook in the chain or to the application
FALSE The message is passed on to the next hook in the chain or to the application.

InputHook - Parameters

hab (**HAB**) - input
Anchor-block handle.

pQmsg (**PQMSG**) - input
A PQMSG data structure.

fs (**ULONG**) - input
Message removal options.

PM_REMOVE Message is being removed from queue
PM_NOREMOVE Message is not being removed from queue.

rc (**BOOL**) - returns
Processed indicator.

TRUE The message is not passed on to the next hook in the chain or to the application
FALSE The message is passed on to the next hook in the chain or to the application.

InputHook - Remarks

This hook is called when messages are removed from an application queue, before being returned by [WinGetMsg](#) or [WinPeekMsg](#). It is called from within these functions just before resuming the application with the message that is returned. There are no restrictions on calls that may be made at this time.

InputHook - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

JournalPlaybackHook

JournalPlaybackHook - Syntax

This hook plays back recorded messages.

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;      /* Anchor-block handle. */
BOOL     fSkip;    /* Indicator as to whether the next message should be played back. */
PQMSG    pQmsg;    /* Data structure where the message to be played back is returned. */
ULONG    ulTime;   /* Waiting time. */

ulTime = JournalPlaybackHook(hab, fSkip, pQmsg);
```

JournalPlaybackHook Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

JournalPlaybackHook Parameter - fSkip

fSkip ([BOOL](#)) - input
Indicator as to whether the next message should be played back.

TRUE

The journal playback hook skips to the next message. The *pQmsg* parameter is NULL in this case. The next hook in the chain is not called.

FALSE

The journal playback hook returns the next available message. The same message is returned each time, until it is skipped with a call where this parameter is TRUE.

JournalPlaybackHook Parameter - pQmsg

pQmsg ([PQMSG](#)) - input

Data structure where the message to be played back is returned.

When this hook is called, the field of the [QMSG](#) structure is initialized to the current time. This can be used to determine whether the next message is ready or not. This value must be used for any delta calculations performed by the hook procedure, rather than the result of [WinGetCurrentTime](#).

JournalPlaybackHook Return Value - ulTime

ulTime ([ULONG](#)) - returns

Waiting time.

The time to wait (in milliseconds) before processing the current message.

JournalPlaybackHook - Parameters

hab ([HAB](#)) - input

Anchor-block handle.

fSkip ([BOOL](#)) - input

Indicator as to whether the next message should be played back.

TRUE

The journal playback hook skips to the next message. The *pQmsg* parameter is NULL in this case. The next hook in the chain is not called.

FALSE

The journal playback hook returns the next available message. The same message is returned each time, until it is skipped with a call where this parameter is TRUE.

pQmsg ([PQMSG](#)) - input

Data structure where the message to be played back is returned.

When this hook is called, the field of the [QMSG](#) structure is initialized to the current time. This can be used to determine whether the next message is ready or not. This value must be used for any delta calculations performed by the hook procedure, rather than the result of [WinGetCurrentTime](#).

ulTime ([ULONG](#)) - returns

Waiting time.

The time to wait (in milliseconds) before processing the current message.

JournalPlaybackHook - Remarks

This hook is called whenever a message is required to be played back.

JournalPlaybackHook - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

JournalRecordHook

JournalRecordHook - Syntax

This hook records user-input messages.

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */  
#include <os2.h>  
  
HAB      hab;      /* Anchor-block handle. */  
PQMSG    pQmsg;    /* Data structure that contains the message to be recorded. */  
  
JournalRecordHook(hab, pQmsg);
```

JournalRecordHook Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

JournalRecordHook Parameter - pQmsg

pQmsg ([PQMSG](#)) - input
Data structure that contains the message to be recorded.

The field of the [QMSG](#) structure is also set when the hook is called.

JournalRecordHook - Return Value

There is no return value for this hook.

JournalRecordHook - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

pQmsg ([PQMSG](#)) - input
Data structure that contains the message to be recorded.
The field of the [QMSG](#) structure is also set when the hook is called.

There is no return value for this hook.

JournalRecordHook - Remarks

This hook is called *after* raw input is translated to [WM_CHAR](#) or [WM_BUTTON1DBLCLK](#) messages.

The next hook in the chain is always called, and the message is always passed to the application.

[JournalPlaybackHook](#) hook does not receive any input played back by this hook. This prevents feedback situations where input is played back a number of times.

JournalRecordHook - Related Messages

Related Messages

- [WM_CHAR](#)
 - [WM_BUTTON1DBLCLK](#)
-

JournalRecordHook - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Messages](#)
[Glossary](#)

LoaderHook

LoaderHook - Syntax

This hook allows the library and procedure loading and deleting calls to be intercepted.

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
LONG     idContext;    /* Origin of call to hook. */
PSZ      pszLibname;   /* Library name. */
PHLIB    hlib;         /* Pointer to a library handle. */
PSZ      pszProcname;  /* Procedure name. */
PFNWP    wndProc;      /* Window procedure identifier. */
PBOOL    pfSuccess;    /* Success indicator. */
BOOL     rc;           /* Processing indicator. */

rc = LoaderHook(hab, idContext, pszLibname,
                hlib, pszProcname, wndProc, pfSuccess);
```

LoaderHook Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

LoaderHook Parameter - idContext

idContext ([LONG](#)) - input
Origin of call to hook.

```
LHK_DELETEPROC    WinDeleteProcedure
LHK_DELETELIB     WinDeleteLibrary
LHK_LOADPROC      WinLoadProcedure
LHK_LOADLIB       WinLoadLibrary
```

LoaderHook Parameter - pszLibname

pszLibname ([PSZ](#)) - input
Library name.

This is the same as the library name in the *pszLibname* parameter of the [WinLoadLibrary](#) function.

LoaderHook Parameter - hlib

hlib ([PHLIB](#)) - in/out
Pointer to a library handle.

This is the same as the library handle in the *hlibLibhandle* parameter of the [WinLoadProcedure](#) function or the *hlibLibhandle* parameter of the [WinDeleteLibrary](#) function.

If the *idContext* parameter is set to LHK_LOADLIB, then this hook must set the value of this parameter to the handle of the loaded library or to NULLHANDLE if the load fails.

LoaderHook Parameter - pszProcname

pszProcname ([PSZ](#)) - input
Procedure name.

This is the same as the procedure name in the *pszProcname* parameter of the [WinLoadProcedure](#) function.

LoaderHook Parameter - wndProc

wndProc ([PFNWP](#)) - input
Window procedure identifier.

This is the same as the library name in the *pwndproc* parameter of the [WinDeleteProcedure](#) function.

If the *idContext* parameter is set to LHK_LOADPROC, then this hook must set the value of this parameter to the handle of the loaded procedure or to NULL if the load fails.

LoaderHook Parameter - pfSuccess

pfSuccess ([PBOOL](#)) - in/out
Success indicator.

TRUE	Library or procedure loaded or deleted successfully.
FALSE	Library or procedure not loaded or deleted successfully.

LoaderHook Return Value - rc

rc ([BOOL](#)) - returns
Processing indicator.

TRUE	Do not call next hook in chain.
FALSE	Call next hook in chain.

LoaderHook - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

idContext ([LONG](#)) - input
Origin of call to hook.

LHK_DELETEPROC	WinDeleteProcedure
LHK_DELETELIB	WinDeleteLibrary
LHK_LOADPROC	WinLoadProcedure
LHK_LOADLIB	WinLoadLibrary

pszLibname ([PSZ](#)) - input
Library name.

This is the same as the library name in the *pszLibname* parameter of the [WinLoadLibrary](#) function.

hlib ([PHLIB](#)) - in/out
Pointer to a library handle.

This is the same as the library handle in the *hlibLibhandle* parameter of the [WinLoadProcedure](#) function or the *hlibLibhandle* parameter of the [WinDeleteLibrary](#) function.

If the *idContext* parameter is set to LHK_LOADLIB, then this hook must set the value of this parameter to the handle of the loaded library or to NULLHANDLE if the load fails.

pszProcname ([PSZ](#)) - input
Procedure name.

This is the same as the procedure name in the *pszProcname* parameter of the [WinLoadProcedure](#) function.

wndProc ([PFNWP](#)) - input
Window procedure identifier.

This is the same as the library name in the *pwndproc* parameter of the [WinDeleteProcedure](#) function.

If the *idContext* parameter is set to LHK_LOADPROC, then this hook must set the value of this parameter to the handle of the loaded

procedure or to NULL if the load fails.

pfSuccess ([PBOOL](#)) - in/out
Success indicator.

TRUE	Library or procedure loaded or deleted successfully.
FALSE	Library or procedure not loaded or deleted successfully.

rc ([BOOL](#)) - returns
Processing indicator.

TRUE	Do not call next hook in chain.
FALSE	Call next hook in chain.

LoaderHook - Remarks

If the hook attempts a load or deletion which is unsuccessful, then the hook must establish the relevant error information.

LoaderHook - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

LockupHook

LockupHook - Syntax

This function is specific to OS/2 Version 2.1 or higher.

This hook is called when the system locks itself up.

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* The application anchor block. */
HWND     hwndLockupFrame; /* The frame window of the lockup panel. */

LockupHook(hab, hwndLockupFrame);
```

LockupHook Parameter - hab

hab ([HAB](#)) - input
The application anchor block.

LockupHook Parameter - hwndLockupFrame

hwndLockupFrame ([HWND](#)) - input
The frame window of the lockup panel.

LockupHook - Return Value

There is no return value for this hook.

LockupHook - Parameters

hab ([HAB](#)) - input
The application anchor block.

hwndLockupFrame ([HWND](#)) - input
The frame window of the lockup panel.

There is no return value for this hook.

LockupHook - Remarks

The *hwndLockupFrame* parameter contains the frame window handle of the lockup dialog. All HK_LOCKUP hooks registered with the system are called when the system locks itself up. All HK_LOCKUP hooks must be system hooks, not message queue hooks.

Application programs that create other lockup password input windows by hooking the HK_LOCKUP system hook can use [WinUnlockSystem](#) to force the system to unlock when another form of input is detected other than mouse or keyboard. For example, a pen gesture or some voice input or signature recognition.

LockupHook - Related Functions

Related Functions

- [WinLockupSystem](#)
- [WinUnlockSystem](#)

LockupHook - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

MsgControlHook

MsgControlHook - Syntax

This hook allows the call to determine the flow of messages to be intercepted.

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;           /* Anchor-block handle. */
LONG     idContext;     /* Origin of call to hook. */
HWND     hwnd;         /* Window handle. */
PSZ      pszClassName; /* Window class name. */
ULONG    usMsgClass;    /* Message class. */
LONG     idControl;     /* Control setting. */
PBOOL    fSuccess;      /* Success indicator. */
BOOL     rc;           /* Processing indicator. */

rc = MsgControlHook(hab, idContext, hwnd,
    pszClassName, usMsgClass, idControl,
    fSuccess);
```

MsgControlHook Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

MsgControlHook Parameter - idContext

idContext ([LONG](#)) - input
Origin of call to hook.

MCHK_CLASSMSGINTEREST
[WinSetClassMsgInterest](#)
MCHK_MSGINTEREST
[WinSetMsgInterest](#)
MCHK_MSGMODE
[WinSetMsgMode](#)
MCHK_SYNCHRONISATION
[WinSetSynchroMode](#)

MsgControlHook Parameter - hwnd

hwnd ([HWND](#)) - input
Window handle.

This is the same as the window handle in the *hwnd* parameter of the [WinSetMsgInterest](#) function.

MsgControlHook Parameter - pszClassName

pszClassName ([PSZ](#)) - input
Window class name.

This is the same as the window class name in the *pszClassName* parameter of the [WinSetClassMsgInterest](#) function.

MsgControlHook Parameter - usMsgClass

usMsgClass ([ULONG](#)) - input
Message class.

This is the same as the message class in the *ulMsgClass* parameter of the [WinSetMsgInterest](#) and the [WinSetClassMsgInterest](#) functions.

MsgControlHook Parameter - idControl

idControl (**LONG**) - input
Control setting.

The setting varies with the value of the *idContext* parameter.

For MCHK_CLASSMSGINTEREST, it can be SMI_INTEREST, or SMI_NOINTEREST, or SMI_AUTODISPATCH.

For MCHK_MSGINTEREST, it can be SMI_INTEREST, or SMI_NOINTEREST, or SMI_RESET, or SMI_AUTODISPATCH.

For MCHK_MSGMODE, it can be SMD_DELAYED or SMD_IMMEDIATE.

For MCHK_SYNCHRONISATION, it can be SSM_SYNCHRONOUS, or SSM_ASYNCHRONOUS, or SSM_MIXED.

MsgControlHook Parameter - fSuccess

fSuccess (**PBOOL**) - in/out
Success indicator.

TRUE	Mode or interest successfully set.
FALSE	Mode or interest not successfully set.

MsgControlHook Return Value - rc

rc (**BOOL**) - returns
Processing indicator.

TRUE	Do not call next hook in chain
FALSE	Call next hook in chain.

MsgControlHook - Parameters

hab (**HAB**) - input
Anchor-block handle.

idContext (**LONG**) - input
Origin of call to hook.

MCHK_CLASSMSGINTEREST	WinSetClassMsgInterest
MCHK_MSGINTEREST	WinSetMsgInterest
MCHK_MSGMODE	WinSetMsgMode
MCHK_SYNCHRONISATION	

WinSetSynchroMode

hwnd ([HWND](#)) - input
Window handle.

This is the same as the window handle in the *hwnd* parameter of the [WinSetMsgInterest](#) function.

pszClassName ([PSZ](#)) - input
Window class name.

This is the same as the window class name in the *pszClassName* parameter of the [WinSetClassMsgInterest](#) function.

usMsgClass ([ULONG](#)) - input
Message class.

This is the same as the message class in the *uiMsgClass* parameter of the [WinSetMsgInterest](#) and the [WinSetClassMsgInterest](#) functions.

idControl ([LONG](#)) - input
Control setting.

The setting varies with the value of the *idContext* parameter.

For MCHK_CLASSMSGINTEREST, it can be SMI_INTEREST, or SMI_NOINTEREST, or SMI_AUTODISPATCH.

For MCHK_MSGINTEREST, it can be SMI_INTEREST, or SMI_NOINTEREST, or SMI_RESET, or SMI_AUTODISPATCH.

For MCHK_MSGMODE, it can be SMD_DELAYED or SMD_IMMEDIATE.

For MCHK_SYNCHRONISATION, it can be SSM_SYNCHRONOUS, or SSM_ASYNCHRONOUS, or SSM_MIXED.

fSuccess ([PBOOL](#)) - in/out
Success indicator.

TRUE	Mode or interest successfully set.
FALSE	Mode or interest not successfully set.

rc ([BOOL](#)) - returns
Processing indicator.

TRUE	Do not call next hook in chain
FALSE	Call next hook in chain.

MsgControlHook - Remarks

If the hook is unable to alter the message control state, then the hook must establish the relevant error information.

MsgControlHook - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

MsgFilterHook

MsgFilterHook - Syntax

This hook filters messages from inside a mode loop.

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;      /* Anchor-block handle. */
PQMSG    pQmsg;    /* A queue message data structure. */
ULONG    msgf;     /* Context in which the hook has been called. */
BOOL     rc;       /* Processed indicator. */

rc = MsgFilterHook(hab, pQmsg, msgf);
```

MsgFilterHook Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

MsgFilterHook Parameter - pQmsg

pQmsg (**PQMSG**) - input
A queue message data structure.

MsgFilterHook Parameter - msgf

msgf (**ULONG**) - input
Context in which the hook has been called.

MSGF_DIALOGBOX
Dialog-box mode loop.

MSGF_TRACK
Window-movement and size tracking. When this hook is used the **TRACKINFO** structure specified the *ptiTrackinfo* parameter of the **WinTrackRect** function is updated to give the current state before the hook is called. Only the

parameters are updated.

MSGF_DRAG

Direct manipulation mode loop.

MSGF_DDEPOSTMSG

DDE post message mode loop.

MsgFilterHook Return Value - rc

rc (**BOOL**) - returns

Processed indicator.

TRUE

The message is not passed on to the next hook in the chain or to the application

FALSE

The message is passed on to the next hook in the chain or to the application.

MsgFilterHook - Parameters

hab (**HAB**) - input

Anchor-block handle.

pQmsg (**PQMSG**) - input

A queue message data structure.

msgf (**ULONG**) - input

Context in which the hook has been called.

MSGF_DIALOGBOX

Dialog-box mode loop.

MSGF_TRACK

Window-movement and size tracking. When this hook is used the **TRACKINFO** structure specified the *ptTrackinfo* parameter of the **WinTrackRect** function is updated to give the current state before the hook is called. Only the parameters are updated.

MSGF_DRAG

Direct manipulation mode loop.

MSGF_DDEPOSTMSG

DDE post message mode loop.

rc (**BOOL**) - returns

Processed indicator.

TRUE

The message is not passed on to the next hook in the chain or to the application

FALSE

The message is passed on to the next hook in the chain or to the application.

MsgFilterHook - Remarks

This hook is called inside any of the system-mode loops, for instance, during size-tracking or move-tracking, or while a dialog box or menu is displayed.

The [WM_QUIT](#) message is passed to this hook, if it occurs during a mode loop.

MsgFilterHook - Related Messages

Related Messages

- [WM_QUIT](#)

MsgFilterHook - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Related Messages](#)
 - [Glossary](#)

MsgInputHook

MsgInputHook - Syntax

This function is specific to OS/2 Version 2.1 or higher.

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
PQMSG    pQmsg;        /* Queue message data structure. */
BOOL      fSkip;       /* Skip message flag. */
PBOOL    pfNoRecord;   /* Record message flag. */
BOOL      rc;          /* Return code. */

rc = MsgInputHook(hab, pQmsg, fSkip, pfNoRecord);
```

MsgInputHook Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

MsgInputHook Parameter - pQmsg

pQmsg ([PQMSG](#)) - in/out
Queue message data structure.

A queue message data structure to be filled in with a simulated mouse or keyboard message.

MsgInputHook Parameter - fSkip

fSkip ([BOOL](#)) - input
Skip message flag.

TRUE

The hook should advance to the next message.

No message is passed in when *fSkip* is set to TRUE. The *pQmsg* parameter is NULL in this case.

When the hook is called with *fSkip* set to TRUE and there are no further messages to pass in, the MsgInputHook hook must be released using [WinReleaseHook](#).

FALSE

The current message should be returned.

As long as the hook procedure is called with *fSkip* set to FALSE, the hook must continue to return the *same* message.

MsgInputHook Parameter - pfNoRecord

pfNoRecord ([PBOOL](#)) - output
Record message flag.

TRUE

The message will not be recorded in the [JournalRecordHook](#) hook.

Unless journal recording is specifically needed by the application, this parameter should always be TRUE.

FALSE

The message will be recorded in the [JournalRecordHook](#) hook.

MsgInputHook Return Value - rc

rc ([BOOL](#)) - returns
Return code.

TRUE

The [QMSG](#) structure contains the current message to be passed in for handling.

FALSE

The [QMSG](#) structure was not filled in. There are no further messages to pass in.

MsgInputHook - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

pQmsg ([PQMSG](#)) - in/out
Queue message data structure.

A queue message data structure to be filled in with a simulated mouse or keyboard message.

fSkip ([BOOL](#)) - input
Skip message flag.

TRUE

The hook should advance to the next message.

No message is passed in when *fSkip* is set to TRUE. The *pQmsg* parameter is NULL in this case.

When the hook is called with *fSkip* set to TRUE and there are no further messages to pass in, the MsgInputHook hook must be released using [WinReleaseHook](#).

FALSE

The current message should be returned.

As long as the hook procedure is called with *fSkip* set to FALSE, the hook must continue to return the *same* message.

pfNoRecord ([PBOOL](#)) - output
Record message flag.

TRUE

The message will not be recorded in the [JournalRecordHook](#) hook.

Unless journal recording is specifically needed by the application, this parameter should always be TRUE.

FALSE

The message will be recorded in the [JournalRecordHook](#) hook.

rc ([BOOL](#)) - returns
Return code.

TRUE

The [QMSG](#) structure contains the current message to be passed in for handling.

FALSE

The [QMSG](#) structure was not filled in. There are no further messages to pass in.

MsgInputHook - Remarks

This hook is intended for simulated user input, and only mouse and keyboard messages should be passed in. All other messages will be discarded. Mouse and keyboard messages injected into this hook will have the same effect as if they were generated by the mouse or keyboard device driver. The messages are routed in the same manner as normal user input.

The *pfNoRecord* parameter defaults to TRUE, implying that messages received from the MsgInputHook hook will not be recorded in the [JournalRecordHook](#) hook. Injected messages should not be recorded unless necessary.

When all messages have been passed in the application must remove the MsgInputHook hook using [WinReleaseHook](#).

MsgInputHook - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

ProgramListEntryHook

ProgramListEntryHook - Syntax

This hook chain is called every time a program-list call or initialization file call is invoked by an application. It is called before the call is run.

```
#define INCL_FnDrEnPt
#include <os2.h>

HAB      hab;
PRFHOOKPARMS pProfileHookParams;
BOOL     fNoExecute;
BOOL     rc;

rc = ProgramListEntryHook(hab, pProfileHookParams,
    fNoExecute);
```

ProgramListEntryHook Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

ProgramListEntryHook Parameter - pProfileHookParams

pProfileHookParams ([PRFHOOKPARMS](#)) - input

Profile hook parameters. These identify the call and give its parameters and return code. See [PRFHOOKPARMS](#) for more information.

ProgramListEntryHook Parameter - fNoExecute

fNoExecute ([BOOL](#)) - output

Suppress indicator.

TRUE

If this is set by any hook procedure, no further processing of the call is done.

FALSE

If all hook procedures set *fNoExecute* to FALSE, the call is processed normally.

ProgramListEntryHook Return Value - rc

rc ([BOOL](#)) - returns

Processed indicator.

TRUE

The next hook in the chain is not called.

FALSE

The next hook in the chain is called.

ProgramListEntryHook - Parameters

hab ([HAB](#)) - input

Anchor-block handle.

pProfileHookParams ([PRFHOOKPARMS](#)) - input

Profile hook parameters. These identify the call and give its parameters and return code. See [PRFHOOKPARMS](#) for more information.

fNoExecute ([BOOL](#)) - output

Suppress indicator.

TRUE

If this is set by any hook procedure, no further processing of the call is done.

FALSE

If all hook procedures set *fNoExecute* to FALSE, the call is processed normally.

rc ([BOOL](#)) - returns

Processed indicator.

TRUE

FALSE	The next hook in the chain is not called.
	The next hook in the chain is called.

ProgramListEntryHook - Remarks

This hook, together with [ProgramListExitHook](#) lets applications or system components to do the following:

1. Implement the initialization file and program list partially, while retaining the existing implementation. For example, read-only requests could be satisfied from memory, rather than from disk.
2. Redirect initialization-file operations on a particular group to an alternative (opened) profile. For example, in a multiple-user environment, a LAN program might choose to redirect profile groups that are hardware-dependent, rather than user-dependent, to the system-initialization file.

ProgramListEntryHook - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

ProgramListExitHook

ProgramListExitHook - Syntax

This hook chain is called every time a program-list call or initialization file call is invoked by an application. It is called before the call is run.

```
#define INCL_FnDrEnPt
#include <os2.h>

HAB          hab;
PRFHOOKPARMS pProfileHookParams;
BOOL         rc;

rc = ProgramListExitHook(hab, pProfileHookParams);
```

ProgramListExitHook Parameter - hab

hab ([HAB](#)) - input
Anchor block handle.

ProgramListExitHook Parameter - pProfileHookParams

pProfileHookParams ([PRFHOOKPARMS](#)) - input
Profile hook parameters. These identify the call and give its parameters and return code. See [PRFHOOKPARMS](#).

ProgramListExitHook Return Value - rc

rc ([BOOL](#)) - returns
Process indicator.

TRUE	The next hook in the chain is not called.
FALSE	The next hook in the chain is called.

ProgramListExitHook - Parameters

hab ([HAB](#)) - input
Anchor block handle.

pProfileHookParams ([PRFHOOKPARMS](#)) - input
Profile hook parameters. These identify the call and give its parameters and return code. See [PRFHOOKPARMS](#).

rc ([BOOL](#)) - returns
Process indicator.

TRUE	The next hook in the chain is not called.
FALSE	The next hook in the chain is called.

ProgramListExitHook - Remarks

This hook, together with its counterpart, [ProgramListEntryHook](#), lets applications or system components:

1. Implement the initialization file and program list partially, whilst retaining the existing implementation. For example, read-only requests could be satisfied from memory, rather than from disk.
2. Redirect initialization-file operations on a particular group to an alternative (opened) profile. For example, in a multiple-user

environment, a LAN program might choose to redirect profile groups that are hardware-dependent, rather than user-dependent, to the system-initialization file.

ProgramListExitHook - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

RegisterUserHook

RegisterUserHook - Syntax

This hook is called whenever a user message or data type is registered.

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* The application anchor block. */
ULONG    cUshort;      /* Number of data type codes. */
PUSHORT  arRMP;        /* Array of data type codes. */
PBOOL    fRegistered;  /* Flag indicating that a message or data type was registered. */
BOOL     rc;           /* Success indicator. */

rc = RegisterUserHook(hab, cUshort, arRMP,
    fRegistered);
```

RegisterUserHook Parameter - hab

hab ([HAB](#)) - input
The application anchor block.

RegisterUserHook Parameter - cUshort

cUshort ([ULONG](#)) - input
Number of data type codes.

For data types, this parameter is the number of data type codes in *arRMP*, This value must not be less than one.

For messages, this parameter is set to 0.

RegisterUserHook Parameter - arRMP

arRMP ([PUSHORT](#)) - input
Array of data type codes.

For data types, this parameter is an array of data type codes. For messages, this parameter is set to NULL.

Valid data types are the system-defined data types and their pointer equivalents, application-defined data types and their pointer equivalents, and control data types. Note that not all of the data types that occur in the CPI can be specified in this function.

A control data type is followed by one or more entries in the *arRMP* array that are interpreted in a special way. Control data types allow arrays, offsets, and lengths to be defined.

Simple Data Types

DTYP_ATOM	See ATOM .
DTYP_BIT16	See USHORT .
DTYP_BIT32	See ULONG .
DTYP_BIT8	See UCHAR .
DTYP_BOOL	See BOOL .
DTYP_COUNT2	See USHORT .
DTYP_COUNT2B	See USHORT .
DTYP_COUNT2CH	See USHORT .
DTYP_COUNT4B	See ULONG .
DTYP_CPID	See USHORT .
DTYP_ERRORID	See ERRORID .
DTYP_IDENTITY	See USHORT .
DTYP_IDENTITY4	See ULONG .
DTYP_INDEX2	See USHORT .
DTYP_IPT	See IPT .
DTYP_LENGTH2	See USHORT .
DTYP_LENGTH4	See ULONG .
DTYP_LONG	See LONG .
DTYP_OFFSET2B	See USHORT .
DTYP_PID	See PID .
DTYP_PIX	See PIX .
DTYP_PROGCATEGORY	See PROGCATEGORY .
DTYP_PROPERTY2	See USHORT .
DTYP_PROPERTY4	See LONG .
DTYP_RESID	See HMODULE .
DTYP_SEGOFF	See SEGOFF .
DTYP_SHORT	See SHORT .
DTYP_TID	See TID .
DTYP_TIME	See LONG .
DTYP_UCHAR	See UCHAR .
DTYP_ULONG	See ULONG .
DTYP_USHORT	See USHORT .
DTYP_WIDTH4	See LONG .
DTYP_WNDPROC	See PFNWP . <i>Handle Data Types</i>
DTYP_HAB	See HAB .
DTYP_HACCEL	See HACCEL .
DTYP_HAPP	See HAPP .
DTYP_HATOMTBL	See HATOMTBL .
DTYP_HBITMAP	See HBITMAP .
DTYP_HDC	See HDC .
DTYP_HENUM	See HENUM .
DTYP_HINI	See HINI .

DTYP_HLIB
DTYP_HMF
DTYP_HMQ
DTYP_HPOINTER
DTYP_HPROGRAM
DTYP_HPS
DTYP_HRGN
DTYP_HSEM
DTYP_HSPL
DTYP_HSWITCH
DTYP_HWND

See [HLIB](#).
See [HMF](#).
See [HMQ](#).
See [HPOINTER](#).
See [HPROGRAM](#).
See [HPS](#).
See [HRGN](#).
See [HSEM](#).
See [HSPL](#).
See [HSWITCH](#).
See [HWND](#).

Character/String/Buffer Data Types

DTYP_BYTE
DTYP_CHAR
DTYP_STRL
DTYP_STR16
DTYP_STR32
DTYP_STR64
DTYP_STR8

See [BYTE](#).
See [CHAR](#).
See [PSZ](#).
See [STR16](#).
See [STR32](#).
See [STR64](#).
See [STR8](#).

Structure Data Types

DTYP_ACCEL
DTYP_ACCELTABLE
DTYP_ARCPARAMS
DTYP_AREABUNDLE
DTYP_BITMAPINFO
DTYP_BITMAPINFOHEADER
DTYP_BTNCDATA
DTYP_CATCHBUF
DTYP_CHARBUNDLE
DTYP_CLASSINFO
DTYP_CREATESTRUCT
DTYP_CURSORINFO
DTYP_DEVOPENSTRUC
DTYP_DLGTEMPLATE
DTYP_DLGITEM
DTYP_ENTRYFDATA
DTYP_FATTRS
DTYP_FFDESCS
DTYP_FIXED
DTYP_FONTMETRICS
DTYP_FRAMECDATA
DTYP_GRADIENTL
DTYP_HCINFO
DTYP_IMAGEBUNDLE
DTYP_KERNINGPAIRS
DTYP_LINEBUNDLE
DTYP_MARGSTRUCT
DTYP_MARKERBUNDLE
DTYP_MATRIXLF
DTYP_MLECTLDATA
DTYP_OVERFLOW
DTYP_OWNERITEM
DTYP_POINTERINFO
DTYP_POINTL
DTYP_PROGRAMENTRY
DTYP_PROGTYPE
DTYP_QMSG
DTYP_RECTL
DTYP_RGB
DTYP_RGNRECT
DTYP_SBCDATA
DTYP_SIZEF
DTYP_SIZEL
DTYP_SWBLOCK
DTYP_SWCNTRL
DTYP_SWENTRY
DTYP_SWP
DTYP_TRACKINFO
DTYP_USERBUTTON

See [ACCEL](#).
See [ACCELTABLE](#).
See [ARCPARAMS](#).
See [AREABUNDLE](#).
See [BITMAPINFO](#).
See [BITMAPINFOHEADER](#).
See [BTNCDATA](#).
See [CATCHBUF](#).
See [CHARBUNDLE](#).
See [CLASSINFO](#).
See [CREATESTRUCT](#).
See [CURSORINFO](#).
See [DEVOPENSTRUC](#).
See [DLGTEMPLATE](#).
See [DLGITEM](#).
See [ENTRYFDATA](#).
See [FATTRS](#).
See [FFDESCS](#).
See [FIXED](#).
See [FONTMETRICS](#).
See [FRAMECDATA](#).
See [GRADIENTL](#).
See [HCINFO](#).
See [IMAGEBUNDLE](#).
See [KERNINGPAIRS](#).
See [LINEBUNDLE](#).
See [MLEMARGSTRUCT](#).
See [MARKERBUNDLE](#).
See [MATRIXLF](#).
See [MLECTLDATA](#).
See [MLEOVERFLOW](#).
See [OWNERITEM](#).
See [POINTERINFO](#).
See [POINTL](#).
See [PROGRAMENTRY](#).
See [PROGTYPE](#).
See [QMSG](#).
See [RECTL](#).
See [RGB](#).
See [RGNRECT](#).
See [SBCDATA](#).
See [SIZEF](#).
See [SIZEL](#).
See [SWBLOCK](#).
See [SWCNTRL](#).
See [SWENTRY](#).
See [SWP](#).
See [TRACKINFO](#).
See [USERBUTTON](#).

DTYP_WNDPARAMS
DTYP_WPOINT
DTYP_WRECT
DTYP_XYWINSIZE

See [WNDPARAMS](#).
See [WPOINT](#).
See [WRECT](#).
See [XYWINSIZE](#).

Pointer Data Type

DTYP_P*

Pointer to an item of data type DTYP_*, where DTYP_* is one of the data types in the preceding lists. The value of a pointer data type is the value of the corresponding non-pointer data type prefixed with a minus to make it negative.

Minimum Application Data Type

DTYP_USER

Minimum value for application-defined non-pointer data types.

Control Data Type

DTYP_CTL_ARRAY

This starts a sequence of three array elements that define an array; the array resides in the structure being defined, and may have a fixed number of elements or a variable number of elements.

The following describes the possible elements:
n DTYP_CTL_ARRAY

n+1 Data type of array data.

n+2 Minus the number of elements in the array (for an array of fixed size). This component must have a suitable numeric data type. The index is zero-based.

DTYP_CTL_LENGTH

This starts a sequence of four array elements that define a structure component containing the length of part or all of the structure. The length component resides at this point in the structure.

The following describes the possible elements:
n DYP_CTL_LENGTH

n+1 Data type of structure component that contains the length (must be a suitable numeric data type).

n+2 The index of the element. A value of -1 denotes the start of the structure. This index is zero-based.

The element specified must not be one that is the second or subsequent element in a DTYP_CTL_* sequence of elements.

N+3 The index of the element. It must not be less than the value contained in element n+2. A value of -1 denotes the end of the structure. The index is zero-based.

The element specified must not be one that is the second or subsequent element in a DTYP_CTL_* sequence of elements.

If the value is -1, the length includes all offset data residing at the end of the structure.

DTYP_CTL_OFFSET

This starts a sequence of four array elements that define data addressed by an offset. The offset resides at this point in the structure, and contains the offset in bytes of the data from the start of the *outermost* structure in which this component resides. The data addressed by the offset must occupy storage following the fixed part of the structure. The data may be scalar data or array data.

DTYP_CTL_PARRAY	The following describes the possible elements:	
	n	DTYP_CTL_OFFSET
	n+1	Data type of the structure component that contains the offset (must be a suitable unsigned numeric data type).
	n+2	Data type of offset data.
	n+3	Minus the number of elements in the array (for an array of fixed size). This component must have a suitable numeric data type; the array-size element may occur earlier or later in the structure. This index is zero-based.
		A value of -1 indicates that the data is not an array.
		This starts a sequence of three array elements that define a pointer to an array. The pointer resides at this point in the structure, and the array resides elsewhere. The array can have a fixed or variable number of elements.
		The following describes the possible elements:
		n DTYP_CTL_PARRAY
		n+1 Data type of array data.
		n+2 Minus the number of elements in the array (for an array of fixed size). This component must have a suitable numeric data type. The array-size element may occur earlier or later in the structure. The index is zero-based.

RegisterUserHook Parameter - fRegistered

fRegistered (PBOOL) - input
Flag indicating that a message or data type was registered.

TRUE	Message/data type was registered.
FALSE	Message/data type was not registered.

RegisterUserHook Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Errors occurred.

RegisterUserHook - Parameters

hab ([HAB](#)) - input
The application anchor block.

cUshort ([ULONG](#)) - input
Number of data type codes.

For data types, this parameter is the number of data type codes in *arRMP*. This value must not be less than one.

For messages, this parameter is set to 0.

arRMP ([PUSHORT](#)) - input
Array of data type codes.

For data types, this parameter is an array of data type codes. For messages, this parameter is set to NULL.

Valid data types are the system-defined data types and their pointer equivalents, application-defined data types and their pointer equivalents, and control data types. Note that not all of the data types that occur in the CPI can be specified in this function.

A control data type is followed by one or more entries in the *arRMP* array that are interpreted in a special way. Control data types allow arrays, offsets, and lengths to be defined.

Simple Data Types

DTYP_ATOM	See ATOM .
DTYP_BIT16	See USHORT .
DTYP_BIT32	See ULONG .
DTYP_BIT8	See UCHAR .
DTYP_BOOL	See BOOL .
DTYP_COUNT2	See USHORT .
DTYP_COUNT2B	See USHORT .
DTYP_COUNT2CH	See USHORT .
DTYP_COUNT4B	See ULONG .
DTYP_CPID	See USHORT .
DTYP_ERRORID	See ERRORID .
DTYP_IDENTITY	See USHORT .
DTYP_IDENTITY4	See ULONG .
DTYP_INDEX2	See USHORT .
DTYP_IPT	See IPT .
DTYP_LENGTH2	See USHORT .
DTYP_LENGTH4	See ULONG .
DTYP_LONG	See LONG .
DTYP_OFFSET2B	See USHORT .
DTYP_PID	See PID .
DTYP_PIX	See PIX .
DTYP_PROGCATEGORY	See PROGCATEGORY .
DTYP_PROPERTY2	See USHORT .
DTYP_PROPERTY4	See LONG .
DTYP_RESID	See HMODULE .
DTYP_SEGOFF	See SEGOFF .
DTYP_SHORT	See SHORT .
DTYP_TID	See TID .
DTYP_TIME	See LONG .
DTYP_UCHAR	See UCHAR .
DTYP_ULONG	See ULONG .
DTYP_USHORT	See USHORT .
DTYP_WIDTH4	See LONG .
DTYP_WNDPROC	See PFNWP . <i>Handle Data Types</i>
DTYP_HAB	See HAB .
DTYP_HACCEL	See HACCEL .
DTYP_HAPP	See HAPP .
DTYP_HATOMTBL	See HATOMTBL .
DTYP_HBITMAP	See HBITMAP .
DTYP_HDC	See HDC .
DTYP_HENUM	See HENUM .

DTYP_HINI
DTYP_HLIB
DTYP_HMF
DTYP_HMQ
DTYP_HPOINTER
DTYP_HPROGRAM
DTYP_HPS
DTYP_HRGN
DTYP_HSEM
DTYP_HSPL
DTYP_HSWITCH
DTYP_HWND

See [HINI](#).
See [HLIB](#).
See [HMF](#).
See [HMQ](#).
See [HPOINTER](#).
See [HPROGRAM](#).
See [HPS](#).
See [HRGN](#).
See [HSEM](#).
See [HSPL](#).
See [HSWITCH](#).
See [HWND](#).

Character/String/Buffer Data Types

DTYP_BYTE
DTYP_CHAR
DTYP_STRL
DTYP_STR16
DTYP_STR32
DTYP_STR64
DTYP_STR8

See [BYTE](#).
See [CHAR](#).
See [PSZ](#).
See [STR16](#).
See [STR32](#).
See [STR64](#).
See [STR8](#).

Structure Data Types

DTYP_ACCEL
DTYP_ACCELTABLE
DTYP_ARCPARAMS
DTYP_AREABUNDLE
DTYP_BITMAPINFO
DTYP_BITMAPINFOHEADER
DTYP_BTNCDATA
DTYP_CATCHBUF
DTYP_CHARBUNDLE
DTYP_CLASSINFO
DTYP_CREATESTRUCT
DTYP_CURSORINFO
DTYP_DEVOPENSTRUC
DTYP_DLGTEMPLATE
DTYP_DLGITEM
DTYP_ENTRYFDATA
DTYP_FATTRS
DTYP_FFDESCS
DTYP_FIXED
DTYP_FONTMETRICS
DTYP_FRAMECDATA
DTYP_GRADIENTL
DTYP_HCINFO
DTYP_IMAGEBUNDLE
DTYP_KERNINGPAIRS
DTYP_LINEBUNDLE
DTYP_MARGSTRUCT
DTYP_MARKERBUNDLE
DTYP_MATRIXLF
DTYP_MLECTLDATA
DTYP_OVERFLOW
DTYP_OWNERITEM
DTYP_POINTERINFO
DTYP_POINTL
DTYP_PROGRAMENTRY
DTYP_PROGTYPE
DTYP_QMSG
DTYP_RECTL
DTYP_RGB
DTYP_RGNRECT
DTYP_SBCDATA
DTYP_SIZEF
DTYP_SIZEL
DTYP_SWBLOCK
DTYP_SWCNTRL
DTYP_SWENTRY
DTYP_SWP
DTYP_TRACKINFO

See [ACCEL](#).
See [ACCELTABLE](#).
See [ARCPARAMS](#).
See [AREABUNDLE](#).
See [BITMAPINFO](#).
See [BITMAPINFOHEADER](#).
See [BTNCDATA](#).
See [CATCHBUF](#).
See [CHARBUNDLE](#).
See [CLASSINFO](#).
See [CREATESTRUCT](#).
See [CURSORINFO](#).
See [DEVOPENSTRUC](#).
See [DLGTEMPLATE](#).
See [DLGITEM](#).
See [ENTRYFDATA](#).
See [FATTRS](#).
See [FFDESCS](#).
See [FIXED](#).
See [FONTMETRICS](#).
See [FRAMECDATA](#).
See [GRADIENTL](#).
See [HCINFO](#).
See [IMAGEBUNDLE](#).
See [KERNINGPAIRS](#).
See [LINEBUNDLE](#).
See [MLEMARGSTRUCT](#).
See [MARKERBUNDLE](#).
See [MATRIXLF](#).
See [MLECTLDATA](#).
See [MLEOVERFLOW](#).
See [OWNERITEM](#).
See [POINTERINFO](#).
See [POINTL](#).
See [PROGRAMENTRY](#).
See [PROGTYPE](#).
See [QMSG](#).
See [RECTL](#).
See [RGB](#).
See [RGNRECT](#).
See [SBCDATA](#).
See [SIZEF](#).
See [SIZEL](#).
See [SWBLOCK](#).
See [SWCNTRL](#).
See [SWENTRY](#).
See [SWP](#).
See [TRACKINFO](#).

DTYP_USERBUTTON
DTYP_WNDPARAMS
DTYP_WPOINT
DTYP_WRECT
DTYP_XYWINSIZE

See [USERBUTTON](#).
See [WNDPARAMS](#).
See [WPOINT](#).
See [WRECT](#).
See [XYWINSIZE](#).

Pointer Data Type

DTYP_P*

Pointer to an item of data type DTYP_*, where DTYP_* is one of the data types in the preceding lists. The value of a pointer data type is the value of the corresponding non-pointer data type prefixed with a minus to make it negative.

Minimum Application Data Type

DTYP_USER

Minimum value for application-defined non-pointer data types.

Control Data Type

DTYP_CTL_ARRAY

This starts a sequence of three array elements that define an array; the array resides in the structure being defined, and may have a fixed number of elements or a variable number of elements.

The following describes the possible elements:

- | | |
|-----|---|
| n | DTYP_CTL_ARRAY |
| n+1 | Data type of array data. |
| n+2 | Minus the number of elements in the array (for an array of fixed size). This component must have a suitable numeric data type. The index is zero-based. |

DTYP_CTL_LENGTH

This starts a sequence of four array elements that define a structure component containing the length of part or all of the structure. The length component resides at this point in the structure.

The following describes the possible elements:

- | | |
|-----|--|
| n | DYP_CTL_LENGTH |
| n+1 | Data type of structure component that contains the length (must be a suitable numeric data type). |
| n+2 | The index of the element. A value of -1 denotes the start of the structure. This index is zero-based.

The element specified must not be one that is the second or subsequent element in a DTYP_CTL_* sequence of elements. |
| N+3 | The index of the element. It must not be less than the value contained in element n+2. A value of -1 denotes the end of the structure. The index is zero-based.

The element specified must not be one that is the second or subsequent element in a DTYP_CTL_* sequence of elements.

If the value is -1, the length includes all offset data residing at the end of the structure. |

DTYP_CTL_OFFSET

This starts a sequence of four array elements that define data addressed by an offset. The offset resides at this point in the structure, and contains the offset in bytes of the data from the start of the *outermost* structure in which this component resides. The data addressed by the offset must occupy storage following the fixed part of the structure. The data may be scalar

		data or array data.
		The following describes the possible elements:
DTYP_CTL_OFFSET	n	
	n+1	Data type of the structure component that contains the offset (must be a suitable unsigned numeric data type).
	n+2	Data type of offset data.
	n+3	Minus the number of elements in the array (for an array of fixed size). This component must have a suitable numeric data type; the array-size element may occur earlier or later in the structure. This index is zero-based.
		A value of -1 indicates that the data is not an array.
DTYP_CTL_PARRAY		This starts a sequence of three array elements that define a pointer to an array. The pointer resides at this point in the structure, and the array resides elsewhere. The array can have a fixed or variable number of elements.
		The following describes the possible elements:
DTYP_CTL_PARRAY	n	
	n+1	Data type of array data.
	n+2	Minus the number of elements in the array (for an array of fixed size). This component must have a suitable numeric data type. The array-size element may occur earlier or later in the structure. The index is zero-based.
fRegistered (PBOOL) - input		Flag indicating that a message or data type was registered.
TRUE		
	Message/data type was registered.	
FALSE		
	Message/data type was not registered.	
rc (BOOL) - returns		Success indicator.
TRUE		
	Successful completion	
FALSE		
	Errors occurred.	

RegisterUserHook - Remarks

This hook is called when a call is made to either [WinRegisterUserDatatype](#) or [WinRegisterUserMsg](#).

RegisterUserHook - Topics

Select an item:
[Syntax](#)

[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

SendMsgHook

SendMsgHook - Syntax

This hook filters messages sent by the [WinSendMsg](#) function.

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
PSMHSTRUCT psmh;       /* Pointer to a send message hook structure. */
BOOL      fInterTask;  /* Intertask indicator. */

SendMsgHook(hab, psmh, fInterTask);
```

SendMsgHook Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

SendMsgHook Parameter - psmh

psmh ([PSMHSTRUCT](#)) - input
Pointer to a send message hook structure.

This a structure contains the parameters to the [WinSendMsg](#) function.

SendMsgHook Parameter - fInterTask

fInterTask ([BOOL](#)) - input
Intertask indicator.

TRUE	
FALSE	The message is sent between tasks (intertask).
	The message is sent within a task (intratask).

SendMessageHook - Return Value

There is no return value for this hook.

SendMessageHook - Parameters

hAb ([HAB](#)) - input
Anchor-block handle.

psmh ([PSMHSTRUCT](#)) - input
Pointer to a send message hook structure.

This structure contains the parameters to the [WinSendMessage](#) function.

fIntertask ([BOOL](#)) - input
Intertask indicator.

TRUE	
FALSE	The message is sent between tasks (intertask).
	The message is sent within a task (intratask).

There is no return value for this hook.

SendMessageHook - Remarks

This hook may be called whenever a window procedure is called via the [WinSendMessage](#) function.

It is called in the context of the sender, whereby if the sender has a queue hook installed it is called, but if the receiver has a queue hook installed it is not called.

The next hook in the chain is always called.

SendMessageHook - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)

ThunkProc

ThunkProc - Syntax

This procedure provides pointer conversion for application-defined messages.

```
#define INCL_NONE
#include <os2.h>

HWND      hwnd;      /* Window handle. */
USHORT    usmsg;      /* Message identity. */
MPARAM    mpParam1;   /* Message parameter 1. */
MPARAM    mpParam2;   /* Message parameter 2. */
PFNWP     pWndProc;   /* Window-procedure identifier. */
MRESULT    mresReply; /* Message-return data. */

mresReply = ThunkProc(hwnd, usmsg, mpParam1,
                      mpParam2, pWndProc);
```

ThunkProc Parameter - hwnd

hwnd (**HWND**) - input
Window handle.

ThunkProc Parameter - usmsg

usmsg (**USHORT**) - input
Message identity.

This is an application-defined message. The value is greater than or equal to WM_USER.

ThunkProc Parameter - mpParam1

mpParam1 (**MPARAM**) - input
Message parameter 1.

ThunkProc Parameter - mpParam2

mpParam2 (**MPARAM**) - input
Message parameter 2.

ThunkProc Parameter - pWndProc

pWndProc (**PFNWP**) - input
Window-procedure identifier.

ThunkProc Return Value - mresReply

mresReply (**MRESULT**) - returns
Message-return data.

ThunkProc - Parameters

hwnd (**HWND**) - input
Window handle.

usmsg (**USHORT**) - input
Message identity.

This is an application-defined message. The value is greater than or equal to WM_USER.

mpParam1 (**MPARAM**) - input
Message parameter 1.

mpParam2 (**MPARAM**) - input
Message parameter 2.

pWndProc (**PFNWP**) - input
Window-procedure identifier.

mresReply (**MRESULT**) - returns
Message-return data.

ThunkProc - Remarks

Pointer conversion is normally performed automatically by the operating system. An application needs to provide its own pointer-conversion procedures only for application-defined messages which may be passed from 16-bit code to 32-bit code.

A pointer-conversion procedure is associated with a window by the [WinSetWindowThunkProc](#) and [WinSetClassThunkProc](#) functions.

The logic of the pointer-conversion procedure is as follows:

1. Convert each message parameter, if necessary. This may include converting any data structures to which the parameter points.
2. Call the window procedure referenced by the *pWndProc* parameter, supplying as arguments *hwnd*, *usmsg*, *mpParam1* and *mpParam2*.
3. Collect the return value and, if necessary, convert it.

Note that structures to which the return value might point cannot be converted.

4. Convert any structures referenced by message parameters which might have been modified by the window procedure. Note that the pointer-conversion procedure should ensure that the original memory is still available before converting the structures.

A pointer-conversion procedure should process only those messages that it recognizes. On receiving unrecognized messages, it should set *usmsg* to 0.

ThunkProc - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

WindowDCHook

WindowDCHook - Syntax

This hook is called when a device context is allocated or freed.

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* The application anchor block. */
HDC      hdc;          /* The current device-context handle. */
HWND     hwnd;         /* The current window handle. */
BOOL     flAssociate;  /* Association flag. */
BOOL     rc;           /* Success indicator. */

rc = WindowDCHook(hab, hdc, hwnd, flAssociate);
```

WindowDCHook Parameter - hab

hab (**HAB**) - input
The application anchor block.

WindowDCHook Parameter - hdc

hdc (**HDC**) - input
The current device-context handle.

WindowDCHook Parameter - hwnd

hwnd (**HWND**) - input
The current window handle.

WindowDCHook Parameter - flAssociate

flAssociate (**BOOL**) - input
Association flag.

TRUE	Device context has been allocated.
FALSE	Device context has been freed.

WindowDCHook Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Errors occurred.

WindowDCHook - Parameters

hab ([HAB](#)) - input
The application anchor block.

hdc ([HDC](#)) - input
The current device-context handle.

hwnd ([HWND](#)) - input
The current window handle.

flAssociate ([BOOL](#)) - input
Association flag.

TRUE	Device context has been allocated.
FALSE	Device context has been freed.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Errors occurred.

WindowDCHook - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Glossary](#)

WndProc

WndProc - Syntax

This defines the window procedure provided by an application.

```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND      hwnd;      /* Window handle. */
ULONG     msg;        /* Message identity. */
MPARAM    mp1;        /* Message parameter 1. */
MPARAM    mp2;        /* Message parameter 2. */
MRESULT   mresReply;  /* Message-return data. */

mresReply = WndProc(hwnd, msg, mp1, mp2);
```

WndProc Parameter - hwnd

hwnd ([HWND](#)) - input
Window handle.

WndProc Parameter - msg

msg ([ULONG](#)) - input
Message identity.

WndProc Parameter - mp1

mp1 ([MPARAM](#)) - input
Message parameter 1.

WndProc Parameter - mp2

mp2 ([MPARAM](#)) - input
Message parameter 2.

WndProc Return Value - mresReply

mresReply ([HRESULT](#)) - returns
Message-return data.

WndProc - Parameters

hwnd ([HWND](#)) - input

Window handle.

msg ([ULONG](#)) - input
Message identity.

mp1 ([MPARAM](#)) - input
Message parameter 1.

mp2 ([MPARAM](#)) - input
Message parameter 2.

mresReply ([MRESULT](#)) - returns
Message-return data.

WndProc - Remarks

This procedure is associated with a window by the *pfmWndProc* of the [WinRegisterClass](#) function.

The window procedure typically processes only some of the messages passed to it. Those messages it does not process must be passed on to the [WinDefWindowProc](#) function, which performs the standard window processing for those messages.

WndProc - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

Profile Functions

This section describes functions that an application would use to query or write user-specific initialization files.

PrfCloseProfile

PrfCloseProfile - Syntax

This function indicates that a profile is no longer available for use.

```
#define INCL_WINSHELLDATA /* Or use INCL_WIN, INCL_PM, */  
#include <os2.h>
```

```
HINI    hini; /* Initialization-file handle. */
BOOL    rc;   /* Success indicator. */

rc = PrfCloseProfile(hini);
```

PrfCloseProfile Parameter - hini

hini (**HINI**) - input
Initialization-file handle.

After this function, the handle is no longer valid.

PrfCloseProfile Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

PrfCloseProfile - Parameters

hini (**HINI**) - input
Initialization-file handle.

After this function, the handle is no longer valid.

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

PrfCloseProfile - Remarks

This function cannot be used to close the current user or system initialization files.

PrfCloseProfile - Errors

Possible returns from [WinGetLastError](#)

PMERR_INI_FILE_IS_SYS_OR_USER (0x1124)
User or system initialization file cannot be closed.

PMERR_INVALID_INI_FILE_HANDLE (0x1115)
An invalid initialization-file handle was specified.

PrfCloseProfile - Related Functions

Related Functions

- [PrfOpenProfile](#)

PrfCloseProfile - Example Code

This example calls PrfCloseProfile to close a profile after it has been used.

```
/* Write and read binary data to a profile.  Some return code */
/* checking omitted for brevity.                               */

#define INCL_WINSHELLDATA
#define INCL_WINERRORS
#define INCL_DOSERRORS
#include <os2.h>
#include <stdio.h>
#include <string.h>

#define BUFSIZE 7

INT main(VOID) {

    HAB    hab          = NULLHANDLE;          /* Window handle */
    HINI    hini         = NULLHANDLE;          /* INI file handle */
    PSZ     pszFileName  = "PROFILE.INI";        /* Name of profile */
    BOOL    rc           = FALSE;               /* Return code */
    PSZ     pszAppName   = "MYAPPL.EXE";        /* Application name */
    PSZ     pszKeyName   = "BINARY_DATA";       /* Data key name */
    APIRET  ret          = NO_ERROR;            /* API return code */
    PBYTE   pData        = NULL;               /* Pointer to data buffer */
    ULONG   ulDataSize   = 0;                  /* Size of data to read */
    INT     idx          = 0;                   /* Loop index */

    hab = WinInitialize( 0 );
    hini = PrfOpenProfile( hab, pszFileName ); /* Open INI file */

    /* Allocate Memory for binary data to be written and read */

    ret = DosAllocMem( (PPVOID)&pData, sizeof(BYTE)*80,
                      (ULONG)PAG_COMMIT | PAG_READ | PAG_WRITE );

    /* Initialize binary data and then write it to profile */

    for(idx = 0; idx < BUFSIZE; idx++) {
        pData[idx] = idx+65;
    }

    rc = PrfWriteProfileData( hini, pszAppName, pszKeyName,
```

```

        (PVOID)pData, (ULONG)BUFSIZE );

if(rc == FALSE) {
    printf("PrfWriteProfileData error code: %X\n", WinGetLastError(hab));
    return 1;
}

    /* Put junk in buffer so we can see if read is successful */

for(idx = 0; idx < BUFSIZE; idx++) {
    pData[idx] = idx;
}

    /* Retrieve size of data and then get data */

rc = PrfQueryProfileSize( hini, pszAppName, pszKeyName, &ulDataSize );
rc = PrfQueryProfileData( hini, pszAppName, pszKeyName,
        (PVOID)pData, &ulDataSize);

if(rc==FALSE) {
    printf("PrfQueryProfileData error code: %X\n", WinGetLastError(hab));
    return 1;
}

printf("Profile Data Read:");
for(idx=0; idx < ulDataSize; idx++)
    printf("%c", pData[idx]);
printf("\n");

PrfCloseProfile(hini);        /* Close profile */
DosFreeMem(pData);           /* Free memory */
return NO_ERROR;              /* Phone home <g> */
}

```

PrfCloseProfile - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

PrfOpenProfile

PrfOpenProfile - Syntax

This function indicates that a file is available for use as a profile.

```
#define INCL_WINSHELLDATA /* Or use INCL_WIN, INCL_PM, */
```



```
#include <os2.h>

HAB      hab;           /* Anchor-block handle. */
PSZ      pszFileName;   /* User-profile file name. */
HINI     hini;          /* Initialization-file handle. */

hini = PrfOpenProfile(hab, pszFileName);
```

PrfOpenProfile Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

PrfOpenProfile Parameter - pszFileName

pszFileName ([PSZ](#)) - input
User-profile file name.

This must not be the same as the current user (OS2.INI) or system initialization (OS2SYS.INI) file name.

PrfOpenProfile Return Value - hini

hini ([HINI](#)) - returns
Initialization-file handle.

This handle is used on other calls to manipulate the profile file.

NULLHANDLE	Error occurred
Other	Initialization-file handle.

PrfOpenProfile - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

pszFileName ([PSZ](#)) - input
User-profile file name.

This must not be the same as the current user (OS2.INI) or system initialization (OS2SYS.INI) file name.

hini ([HINI](#)) - returns

Initialization-file handle.

This handle is used on other calls to manipulate the profile file.

NULLHANDLE

Error occurred

Other

Initialization-file handle.

PrfOpenProfile - Remarks

A user profile and a system profile are opened by the system, either at start-up time, or (in the case of the user profile) as a result of a [PrfReset](#) function, and are always available. Their handles are HINI_USERPROFILE and HINI_SYSTEMPROFILE. Applications do not have to open or close the user profile or the system profile.

The handle returned is only valid for the process issuing the PrfOpenProfile function.

The PrfOpenProfile function can be used by an administrator's application that is creating or modifying a profile for a user.

It can also be used to create a back-up profile as follows:

- Use the enumerate form of [PrfQueryProfileData](#) to obtain a list of application names in the profile being backed up.
- Use the enumerate form of [PrfQueryProfileData](#) to obtain a list of key names for each of the application names.
- Use [PrfQueryProfileData](#) for each application-name or key-name pair to read the appropriate data.
- Use [PrfWriteProfileData](#) to write the data into the back-up profile.

PrfOpenProfile - Errors

Possible returns from [WinGetLastError](#)

PMERR_OPENING_INI_FILE (0x1301)

Unable to open initialization file (due to lack of disk space for example).

PMERR_MEMORY_ALLOC (0x1309)

An error occurred during memory management.

PMERR_INI_FILE_IS_SYS_OR_USER (0x1124)

User or system initialization file cannot be closed.

PrfOpenProfile - Related Functions

Related Functions

- [PrfCloseProfile](#)
- [PrfQueryProfileData](#)

PrfOpenProfile - Example Code

This example uses PrfOpenProfile to open and make available a profile for the file "PROFILE.INI".

```
#define INCL_WINSHELLDATA      /* Window Shell functions */
#include <os2.h>

HINI hini;          /* Initialization-file handle */
HAB hab;            /* Anchor-block handle */
char pszFileName[13]; /* User-profile file name */

strcpy(pszFileName, "PROFILE.INI");

hab = WinInitialize( 0 );
hini = PrfOpenProfile( hab,          /* Anchor block handle */
                      pszFileName); /* Profile name */
```

PrfOpenProfile - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

PrfQueryProfile

PrfQueryProfile - Syntax

This function returns a description of the current user and system profiles.

```
#define INCL_WINSHELLDATA /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB hab;          /* Anchor-block handle. */
PPRFPROFILE pprfproProfile; /* Profile names structure. */
BOOL rc;          /* Success indicator. */

rc = PrfQueryProfile(hab, pprfproProfile);
```

PrfQueryProfile Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

PrfQueryProfile Parameter - pprfproProfile

pprfproProfile ([PPRFPROFILE](#)) - in/out
Profile names structure.

The parameters of the [PPRFPROFILE](#) data structure are set to the lengths of the respective file names, even if truncation occurs. If these fields are initialized to 0 by the application, then the parameters are not inspected, and the application can then determine the sizes of the buffers required to hold the names on a second call. Otherwise, the parameters must point to reserved areas of memory, and the parameters must indicate the sizes of those areas.

If the parameter is NULL, then there is no defined user or system profile, respectively.

PrfQueryProfile Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred, or there was insufficient space to record the names, which have been truncated.

PrfQueryProfile - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

pprfproProfile ([PPRFPROFILE](#)) - in/out
Profile names structure.

The parameters of the [PPRFPROFILE](#) data structure are set to the lengths of the respective file names, even if truncation occurs. If these fields are initialized to 0 by the application, then the parameters are not inspected, and the application can then determine the sizes of the buffers required to hold the names on a second call. Otherwise, the parameters must point to reserved areas of memory, and the parameters must indicate the sizes of those areas.

If the parameter is NULL, then there is no defined user or system profile, respectively.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred, or there was insufficient space to record the names, which have been truncated.

PrfQueryProfile - Related Functions

Related Functions

- [PrfReset](#)
-

PrfQueryProfile - Example Code

This example calls PrfQueryProfile to obtain a description of the current user and system profiles, in this case querying the lengths of the user and system profile file names and placing the values in variables.

```
#define INCL_WINSHELLDATA      /* Window Shell functions      */
#include <os2.h>

BOOL fSuccess;                /* success indicator      */
HAB hab;                      /* anchor-block handle    */
PRFPROFILE pprfproProfile; /* Profile names structure */
ULONG ulUserNameLen;         /* length of user file name */
ULONG ulSysNameLen;          /* length of system file name */

/* initialize lengths so that query will return the buffer sizes*/
pprfproProfile.cchUserName = 0L;
pprfproProfile.cchSysName = 0L;

fSuccess = PrfQueryProfile(hab, &pprfproProfile);

if (fSuccess == TRUE)
{
    ulUserNameLen = pprfproProfile.cchUserName;
    ulSysNameLen = pprfproProfile.cchSysName;
}
```

PrfQueryProfile - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

PrfQueryProfileData

PrfQueryProfileData - Syntax

This function returns a string of binary data from the specified profile.

```
#define INCL_WINSHELLDATA /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HINI      hini;          /* Initialization-file handle. */
PSZ       pszApp;        /* Application name. */
PSZ       pszKey;        /* Key name. */
PVOID     pBuffer;       /* Value data. */
PULONG    pBufferMax;    /* Size of value data. */
BOOL      rc;            /* Success indicator. */

rc = PrfQueryProfileData(hini, pszApp, pszKey,
                        pBuffer, pBufferMax);
```

PrfQueryProfileData Parameter - hini

hini (**HINI**) - input
Initialization-file handle.

HINI_PROFILE	Both the user profile and system profile are searched
HINI_USERPROFILE	The user profile is searched
HINI_SYSTEMPROFILE	The system profile is searched
Other	Initialization-file handle.

PrfQueryProfileData Parameter - pszApp

pszApp (**PSZ**) - input
Application name.

The name of the application for which the profile data is required. The name must match exactly with the name stored in the profile. There is no case-independent searching.

If this parameter is NULL, this function enumerates all the application names present in the profile and returns the names as a list in the *pBuffer* parameter. Each application name is terminated with a NULL character and the last name is terminated with two successive NULL characters. In this case, the *pBufferMax* parameter contains the total length of the list excluding the final NULL character.

PrfQueryProfileData Parameter - pszKey

pszKey (**PSZ**) - input
Key name.

The name of the key for which the profile data is required. The name must match exactly with the name stored in the profile. There is

no case-independent searching.

If this parameter is NULL, and if *pszApp* is not equal to NULL, this call enumerates all key names associated with the named application and returns the key names, but not their values, as a list in the *pBuffer* parameter. Each key name is terminated with a NULL character and the last name is terminated with two successive NULL characters. In this case, the *pulBufferMax* parameter contains the total length of the list **excluding** the final NULL character.

PrfQueryProfileData Parameter - pBuffer

pBuffer (**PVOID**) - output
Value data.

A buffer in which the value corresponding to the key name is returned. The returned data is not null terminated, unless the value data is explicitly null terminated within the file. This function handles binary data.

PrfQueryProfileData Parameter - pulBufferMax

pulBufferMax (**PULONG**) - in/out
Size of value data.

This is the size of the buffer specified by the *pBuffer* parameter. If the call is successful, this is overwritten with the number of bytes copied into the buffer. The maximum amount of data that can be returned is 64K bytes.

PrfQueryProfileData Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

PrfQueryProfileData - Parameters

hini (**HINI**) - input
Initialization-file handle.

HINI_PROFILE	Both the user profile and system profile are searched
HINI_USERPROFILE	The user profile is searched
HINI_SYSTEMPROFILE	The system profile is searched

Other

Initialization-file handle.

pszApp ([PSZ](#)) - input
Application name.

The name of the application for which the profile data is required. The name must match exactly with the name stored in the profile. There is no case-independent searching.

If this parameter is NULL, this function enumerates all the application names present in the profile and returns the names as a list in the *pBuffer* parameter. Each application name is terminated with a NULL character and the last name is terminated with two successive NULL characters. In this case, the *pulBufferMax* parameter contains the total length of the list excluding the final NULL character.

pszKey ([PSZ](#)) - input
Key name.

The name of the key for which the profile data is required. The name must match exactly with the name stored in the profile. There is no case-independent searching.

If this parameter is NULL, and if *pszApp* is not equal to NULL, this call enumerates all key names associated with the named application and returns the key names, but not their values, as a list in the *pBuffer* parameter. Each key name is terminated with a NULL character and the last name is terminated with two successive NULL characters. In this case, the *pulBufferMax* parameter contains the total length of the list **excluding** the final NULL character.

pBuffer ([PVOID](#)) - output
Value data.

A buffer in which the value corresponding to the key name is returned. The returned data is not null terminated, unless the value data is explicitly null terminated within the file. This function handles binary data.

pulBufferMax ([PULONG](#)) - in/out
Size of value data.

This is the size of the buffer specified by the *pBuffer* parameter. If the call is successful, this is overwritten with the number of bytes copied into the buffer. The maximum amount of data that can be returned is 64K bytes.

rc ([BOOL](#)) - returns
Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

PrfQueryProfileData - Remarks

This function returns a string of binary data from the profile. The call searches the file for a key matching the name specified by the *pszKey* parameter, under the application heading specified by the *pszApp* parameter.

Enumeration can be performed in exactly the same way as in the [PrfQueryProfileString](#) function. The enumeration returns application or key names irrespective of whether the data concerned is written with the [PrfWriteProfileString](#) function or the [PrfWriteProfileData](#) function.

This function returns data that is written to the file using either the [PrfWriteProfileString](#) function or the [PrfWriteProfileData](#) function.

If the *pszApp* parameter is NULL, this call enumerates all application names and constructs in the *pBuffer* parameter a list of application names. Each application name in the list is terminated with a null character. The last string in the list is terminated with two null characters. This function returns the length of the list, up to, but not including, the final null. If the enumerated application names exceed the available buffer space, the enumerated names are truncated, the enumerated list is not terminated with 2 bytes of zeros, and the *rc* parameter is set to FALSE. In this case, *pszKey* is ignored.

If the *pszApp* parameter is valid and if the *pszKey* is NULL, this function enumerates all key names associated with the *pszApp* parameter by constructing in the *pBuffer* parameter a list of key names. Each key name in the list is terminated with a null character. The last string in the list is terminated with two null characters. This function returns the length of the list, up to, but not including, the final null.

If the enumerated key names exceed the available buffer space, the enumerated names are truncated, the enumerated list is not terminated

with 2 bytes of zeros, and the `/c` parameter is set to FALSE.

The maximum size of data that can be associated with a key name is 64K bytes.

PrfQueryProfileData - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_PARM (0x1303)

A parameter to the function contained invalid data.

PMERR_NOT_IN_IDX (0x1304)

The application name, key-name or program handle was not found.

PMERR_CAN_NOT_CALL_SPOOLER (0x130D)

An error occurred attempting to call the spooler validation routine. This error is not raised if the spooler is not installed.

PrfQueryProfileData - Related Functions

Related Functions

- [PrfQueryProfileSize](#)
- [PrfWriteProfileData](#)

PrfQueryProfileData - Example Code

This example calls `PrfQueryProfileData` to search a specific profile for the value of key "BINARY_DATA" within the application "MYAPPL.EXE" and return the value if found.

```
/* Write and read binary data to a profile.  Some return code */
/* checking omitted for brevity.                                     */

#define INCL_WINSHELLDATA
#define INCL_WINERRORS
#define INCL_DOSERRORS
#include <os2.h>
#include <stdio.h>
#include <string.h>

#define BUFSIZE 7

INT main(VOID) {

    HAB    hab           = NULLHANDLE;          /* Window handle */
    HINI    hini          = NULLHANDLE;          /* INI file handle */
    PSZ     pszFileName   = "PROFILE.INI";        /* Name of profile */
    BOOL     rc            = FALSE;               /* Return code */
    PSZ     pszAppName    = "MYAPPL.EXE";        /* Application name */
    PSZ     pszKeyName     = "BINARY_DATA";       /* Data key name */
    APIRET   ret           = NO_ERROR;            /* API return code */
    PBYTE    pData         = NULL;               /* Pointer to data buffer */
    ULONG    ulDataSize    = 0;                  /* Size of data to read */
    INT      idx           = 0;                  /* Loop index */

    hab = WinInitialize( 0 );
    hini = PrfOpenProfile( hab, pszFileName );    /* Open INI file */
```

```

    /* Allocate Memory for binary data to be written and read */
    ret = DosAllocMem( (PVOID)&pData, sizeof(BYTE)*80,
        (ULONG)PAG_COMMIT | PAG_READ | PAG_WRITE );

    /* Initialize binary data and then write it to profile */
    for(idx = 0; idx < BUFSIZE; idx++) {
        pData[idx] = idx+65;
    }

    rc = PrfWriteProfileData( hini, pszAppName, pszKeyName,
        (PVOID)pData, (ULONG)BUFSIZE );

    if(rc == FALSE) {
        printf("PrfWriteProfileData error code: %X\n", WinGetLastError(hab));
        return 1;
    }

    /* Put junk in buffer so we can see if read is successful */
    for(idx = 0; idx < BUFSIZE; idx++) {
        pData[idx] = idx;
    }

    /* Retrieve size of data and then get data */

    rc = PrfQueryProfileSize( hini, pszAppName, pszKeyName, &ulDataSize );
    rc = PrfQueryProfileData( hini, pszAppName, pszKeyName,
        (PVOID)pData, &ulDataSize);

    if(rc==FALSE) {
        printf("PrfQueryProfileData error code: %X\n", WinGetLastError(hab));
        return 1;
    }

    printf("Profile Data Read:");
    for(idx=0; idx < ulDataSize; idx++)
        printf("%c", pData[idx]);
    printf("\n");

    PrfCloseProfile(hini);      /* Close profile */
    DosFreeMem(pData);         /* Free memory */
    return NO_ERROR;           /* Phone home <g> */
}

```

PrfQueryProfileData - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

PrfQueryProfileInt

PrfQueryProfileInt - Syntax

This function returns an integer value from the specified profile.

```
#define INCL_WINSHELLDATA /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HINI    hini;        /* Initialization-file handle. */
PSZ     pszApp;      /* Application name. */
PSZ     pszKey;       /* Key name. */
LONG    lDefault;     /* Default value. */
LONG    lResult;      /* Key value specified in the initialization file. */

lResult = PrfQueryProfileInt(hini, pszApp,
                             pszKey, lDefault);
```

PrfQueryProfileInt Parameter - hini

hini ([HINI](#)) - input

Initialization-file handle.

HINI_PROFILE

Both the user profile and system profile are searched

HINI_USERPROFILE

The user profile is searched

HINI_SYSTEMPROFILE

The system profile is searched

Other

Initialization-file handle returned by [PrfOpenProfile](#).

PrfQueryProfileInt Parameter - pszApp

pszApp ([PSZ](#)) - input

Application name.

The name of the application for which the profile data is required. The name must match exactly with the name stored in the profile. There is no case-independent searching.

PrfQueryProfileInt Parameter - pszKey

pszKey ([PSZ](#)) - input

Key name.

The name of the key for which the profile data is required. The name must match exactly with the name stored in the profile. There is no case-independent searching.

PrfQueryProfileInt Parameter - IDefault

IDefault ([LONG](#)) - input
Default value.

This value is returned in *//Result*, if the key defined by *pszKey* cannot be found in the initialization file.

PrfQueryProfileInt Return Value - IResult

IResult ([LONG](#)) - returns
Key value specified in the initialization file.

The value of the key specified by *pszKey* in the initialization file.

If the value corresponding to the key is not an integer, *//Result* is 0.

If the key-name value is a series of digits followed by non-numeric characters, *//Result* contains the value of the digits only. For example, "KeyName=102abc" causes the value 102 to appear in *//Result*.

PrfQueryProfileInt - Parameters

hini ([HINI](#)) - input
Initialization-file handle.

HINI_PROFILE	Both the user profile and system profile are searched
HINI_USERPROFILE	The user profile is searched
HINI_SYSTEMPROFILE	The system profile is searched
Other	Initialization-file handle returned by PrfOpenProfile .

pszApp ([PSZ](#)) - input
Application name.

The name of the application for which the profile data is required. The name must match exactly with the name stored in the profile. There is no case-independent searching.

pszKey ([PSZ](#)) - input
Key name.

The name of the key for which the profile data is required. The name must match exactly with the name stored in the profile. There is no case-independent searching.

IDefault ([LONG](#)) - input
Default value.

This value is returned in *//Result*, if the key defined by *pszKey* cannot be found in the initialization file.

IResult ([LONG](#)) - returns

Key value specified in the initialization file.

The value of the key specified by *pszKey* in the initialization file.

If the value corresponding to the key is not an integer, *IResult* is 0.

If the key-name value is a series of digits followed by non-numeric characters, *IResult* contains the value of the digits only. For example, "KeyName=102abc" causes the value 102 to appear in *IResult*.

PrfQueryProfileInt - Remarks

This function returns an integer value from the profile. The call searches the file for a key matching the name specified by the *pszKey* parameter, under the application heading specified by the *pszApp* parameter. When an integer is stored as a text string using the [PrfWriteProfileString](#) function, for example,

```
PrfWriteProfileString(HINI_PROFILE,
                    app, key,
                    _itoa(123,string,radix));
```

the returned value is the number, 123. When querying an INI file for values, the string in the INI file must be NULL terminated for consistent and correct results. The call returns *Default* if the application-name or key-name pair cannot be found.

Note: The search is case-dependent.

PrfQueryProfileInt - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_PARM (0x1303)

A parameter to the function contained invalid data.

PMERR_NOT_IN_IDX (0x1304)

The application name, key-name or program handle was not found.

PMERR_CAN_NOT_CALL_SPOOLER (0x130D)

An error occurred attempting to call the spooler validation routine. This error is not raised if the spooler is not installed.

PrfQueryProfileInt - Related Functions

Related Functions

- [PrfQueryProfileData](#)
- [PrfWriteProfileString](#)

PrfQueryProfileInt - Example Code

This example writes an integer value to the INDY.INI file. It then looks in that profile for the integer value of key "LEMON_PIN" within the

application "INDY.EXE" and returns the value if found; if not found, -1 is returned.

```
/* Some error checking has been omitted for brevity. */

#define INCL_WINSHELLDATA
#define INCL_WINERRORS
#define INCL_DOSERRORS
#include <os2.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

INT main(VOID) {

    HAB    hab          = NULLHANDLE;
    HINI    hini         = NULLHANDLE;
    PSZ     pszFileName  = "INDY.INI";
    BOOL    rc           = TRUE;
    PSZ     pszAppName   = "INDY.EXE";
    PSZ     pszKeyName   = "LEMON_PIN";
    CHAR    pszString[30];
    LONG    lInputVal    = 7734L,
           lOutputVal    = 0L;

    /* Open profile and write integer out */

    hab = WinInitialize( 0 );
    hini = PrfOpenProfile( hab, pszFileName );

    /* Write integer value to profile. Note that we must
       convert the integer to a string before writing it
       with PrfWrite ProfileString */

    rc = PrfWriteProfileString( hini, pszAppName, pszKeyName,
                               _itoa( lInputVal, pszString, 10 ) );

    if(rc == FALSE) {
        printf("PrfWriteProfileString error code: %X\n", WinGetLastError(hab));
        return 1;
    }

    /* Retrieve integer value of string and display it */

    lOutputVal = PrfQueryProfileInt( hini,
                                    pszAppName,
                                    pszKeyName,
                                    -1); /* -1 will be default */

    if(lOutputVal == 1 || lOutputVal == -1)
        printf("%s\n", lOutputVal == -1 ? "No value for key, returned default" :
               "Key is not an integer");
    else
        printf("Integer value read is %u\n", lOutputVal);

    PrfCloseProfile( hini ); /* Close the profile */

    return NO_ERROR; /* Return to thy caller */
}
```

PrfQueryProfileInt - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)

PrfQueryProfileSize

PrfQueryProfileSize - Syntax

This function obtains the size in bytes of the value of a specified key for a specified application in the profile.

```
#define INCL_WINSHELLDATA /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HINI      hini;          /* Initialization-file handle. */
PSZ       pszApp;        /* Application name. */
PSZ       pszKey;        /* Key name. */
PULONG    pulDataLen;    /* Data length. */
BOOL      rc;            /* Success indicator. */

rc = PrfQueryProfileSize(hini, pszApp, pszKey,
                        pulDataLen);
```

PrfQueryProfileSize Parameter - hini

hini ([HINI](#)) - input
Initialization-file handle.

HINI_PROFILE	Both the user profile and system profile are searched
HINI_USERPROFILE	The user profile is searched
HINI_SYSTEMPROFILE	The system profile is searched
Other	Initialization-file handle returned by PrfOpenProfile .

PrfQueryProfileSize Parameter - pszApp

pszApp ([PSZ](#)) - input
Application name.

The name of the application for which the profile data is required.

If the *pszApp* parameter is NULL, then the *pulDataLen* parameter returns the length of the buffer required to hold the enumerated list of application names, as returned by the [PrfQueryProfileString](#) function when its *pszApp* parameter is NULL. In this case, the

pszKey parameter is ignored.

PrfQueryProfileSize Parameter - pszKey

pszKey (**PSZ**) - input
Key name.

The name of the key for which the size of the data is to be returned.

If the *pszKey* parameter is NULL, and if the *pszApp* parameter is not NULL, the *pulDataLen* returns the length of the buffer required to hold the enumerated list of key names for that application name, as returned by the [PrfQueryProfileString](#) function when its *pszKey* parameter is NULL, and its *pszApp* parameter is not NULL.

PrfQueryProfileSize Parameter - pulDataLen

pulDataLen (**PULONG**) - output
Data length.

This parameter is the length of the value data related to the *pszKey* parameter. If an error occurs, this parameter is undefined. The maximum size of data than can be returned is 64K bytes.

PrfQueryProfileSize Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

PrfQueryProfileSize - Parameters

hini (**HINI**) - input
Initialization-file handle.

HINI_PROFILE	Both the user profile and system profile are searched
HINI_USERPROFILE	The user profile is searched
HINI_SYSTEMPROFILE	The system profile is searched
Other	Initialization-file handle returned by PrfOpenProfile .

pszApp (PSZ) - input
Application name.

The name of the application for which the profile data is required.

If the *pszApp* parameter is NULL, then the *pulDataLen* parameter returns the length of the buffer required to hold the enumerated list of application names, as returned by the [PrfQueryProfileString](#) function when its *pszApp* parameter is NULL. In this case, the *pszKey* parameter is ignored.

pszKey (PSZ) - input
Key name.

The name of the key for which the size of the data is to be returned.

If the *pszKey* parameter is NULL, and if the *pszApp* parameter is not NULL, the *pulDataLen* returns the length of the buffer required to hold the enumerated list of key names for that application name, as returned by the [PrfQueryProfileString](#) function when its *pszKey* parameter is NULL, and its *pszApp* parameter is not NULL.

pulDataLen (PULONG) - output
Data length.

This parameter is the length of the value data related to the *pszKey* parameter. If an error occurs, this parameter is undefined. The maximum size of data than can be returned is 64K bytes.

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

PrfQueryProfileSize - Remarks

The *pszApp* parameter and *pszKey* parameter are case sensitive and must match the names stored in the file exactly. There is no case-independent searching.

This function can be used before using the [PrfQueryProfileString](#) call or the [PrfQueryProfileData](#) call, to allocate space for the returned data.

No distinction is made between data that is written using the [PrfWriteProfileData](#) function and the [PrfWriteProfileString](#) function.

The maximum amount of data that can be associated with a key name is 64K bytes.

PrfQueryProfileSize - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_PARM (0x1303)
A parameter to the function contained invalid data.

PMERR_NOT_IN_IDX (0x1304)
The application name, key-name or program handle was not found.

PMERR_CAN_NOT_CALL_SPOOLER (0x130D)
An error occurred attempting to call the spooler validation routine. This error is not raised if the spooler is not installed.

PrfQueryProfileSize - Related Functions

Related Functions

- [PrfQueryProfileData](#)
- [PrfQueryProfileString](#)

PrfQueryProfileSize - Example Code

This example calls PrfQueryProfileSize to search the user and system profiles for the value of key "KEY" within the application "APP" and return the byte size of the value if found.

```
#define INCL_WINSHELLDATA      /* Window Shell functions */
#include <os2.h>

BOOL fSuccess;                /* success indicator */
HINI hini;                    /* initialization-file handle */
char pszApp[10];              /* application name */
char pszKey[10];              /* key name */
ULONG pDataLen;              /* data length */

/* Both the user profile and system profile are searched */
hini = HINI_PROFILE;

/* specify application and key names */
strcpy(pszApp, "APP");
strcpy(pszKey, "KEY");

fSuccess = PrfQueryProfileSize(hini, pszApp, pszKey, &pDataLen);
```

PrfQueryProfileSize - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

PrfQueryProfileString

PrfQueryProfileString - Syntax

This function retrieves a string from the specified profile. This function is designed to search data written with [PrfWriteProfileString](#)

```

#define INCL_WINSHELLDATA /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HINI      hini;           /* Initialization-file handle. */
PSZ       pszApp;         /* Application name. */
PSZ       pszKey;         /* Key name. */
PSZ       pszDefault;     /* Default string. */
PVOID     pBuffer;        /* Profile string. */
ULONG     ulBufferMax;    /* Maximum string length. */
ULONG     ulLength;       /* String length returned. */

ulLength = PrfQueryProfileString(hini, pszApp,
                                pszKey, pszDefault, pBuffer,
                                ulBufferMax);

```

PrfQueryProfileString Parameter - hini

hini (**HINI**) - input
Initialization-file handle.

HINI_PROFILE	Both the user profile and system profile are searched
HINI_USERPROFILE	The user profile is searched
HINI_SYSTEMPROFILE	The system profile is searched
Other	Initialization-file handle returned by the PrfOpenProfile function.

PrfQueryProfileString Parameter - pszApp

pszApp (**PSZ**) - input
Application name.

The name of the application for which the profile data is required.

The search performed on the application name is always case-dependent. Names starting with the characters "PM_" are reserved for system use.

If this parameter is NULL, this function enumerates all the application names present in the profile and returns the names as a list in the *pBuffer* parameter. Each application name is terminated with a NULL character and the last name is terminated with two successive NULL characters. In this instance, the *ulLength* parameter contains the total length of the list **excluding** the final NULL character.

PrfQueryProfileString Parameter - pszKey

pszKey (**PSZ**) - input
Key name.

The name of the key for which the profile data is returned.

The search on key name is always case-dependent.

If this parameter equals NULL, and if the *pszApp* parameter is not equal to NULL, this function enumerates all key names associated with the named application and returns the key names (not their values) as a list in the *pBuffer* parameter. Each key name is terminated with a NULL character and the last name is terminated with two successive NULL characters. In this instance, the *ulLength* parameter contains the total length of the list **excluding** the final NULL character.

PrfQueryProfileString Parameter - pszDefault

pszDefault (**PSZ**) - input
Default string.

The string that is returned in the *pBuffer* parameter, if the key defined by the *pszKey* parameter cannot be found in the profile.

If the pointer to this parameter is passed as NULL, then nothing is copied into the *pszKey* parameter if the key cannot be found. *ulLength* is returned as 0 in this case.

PrfQueryProfileString Parameter - pBuffer

pBuffer (**PVOID**) - output
Profile string.

The text string obtained from the profile for the key defined by the *pszKey* parameter.

PrfQueryProfileString Parameter - ulBufferMax

ulBufferMax (**ULONG**) - input
Maximum string length.

The maximum number of characters that can be put into the *pBuffer* parameter, in bytes. If the data from the profile is longer than this, it is truncated.

PrfQueryProfileString Return Value - ulLength

ulLength (**ULONG**) - returns
String length returned.

The actual number of characters (including the null termination character) returned in the *pBuffer* parameter, in bytes.

PrfQueryProfileString - Parameters

hini (**HINI**) - input
Initialization-file handle.

HINI_PROFILE
Both the user profile and system profile are searched
HINI_USERPROFILE
The user profile is searched
HINI_SYSTEMPROFILE
The system profile is searched
Other
Initialization-file handle returned by the PrfOpenProfile function.

pszApp (**PSZ**) - input
Application name.

The name of the application for which the profile data is required.

The search performed on the application name is always case-dependent. Names starting with the characters "PM_" are reserved for system use.

If this parameter is NULL, this function enumerates all the application names present in the profile and returns the names as a list in the *pBuffer* parameter. Each application name is terminated with a NULL character and the last name is terminated with two successive NULL characters. In this instance, the *ulLength* parameter contains the total length of the list **excluding** the final NULL character.

pszKey (**PSZ**) - input
Key name.

The name of the key for which the profile data is returned.

The search on key name is always case-dependent.

If this parameter equals NULL, and if the *pszApp* parameter is not equal to NULL, this function enumerates all key names associated with the named application and returns the key names (not their values) as a list in the *pBuffer* parameter. Each key name is terminated with a NULL character and the last name is terminated with two successive NULL characters. In this instance, the *ulLength* parameter contains the total length of the list **excluding** the final NULL character.

pszDefault (**PSZ**) - input
Default string.

The string that is returned in the *pBuffer* parameter, if the key defined by the *pszKey* parameter cannot be found in the profile.

If the pointer to this parameter is passed as NULL, then nothing is copied into the *pszKey* parameter if the key cannot be found. *ulLength* is returned as 0 in this case.

pBuffer (**PVOID**) - output
Profile string.

The text string obtained from the profile for the key defined by the *pszKey* parameter.

ulBufferMax (**ULONG**) - input
Maximum string length.

The maximum number of characters that can be put into the *pBuffer* parameter, in bytes. If the data from the profile is longer than this, it is truncated.

ulLength (**ULONG**) - returns
String length returned.

The actual number of characters (including the null termination character) returned in the *pBuffer* parameter, in bytes.

PrfQueryProfileString - Remarks

The call searches the profile for a key matching the name specified by the *pszKey* parameter under the application heading specified by the *pszApp* parameter. If the key is found, the corresponding string is copied. If the key does not exist, the default character string, specified by the *pszDefault* parameter, is copied.

If the enumerated application names exceed the available buffer space, the enumerated names are truncated, the enumerated list is not terminated with 2 bytes of zeros, and the *ulLength* parameter is set to the number of bytes copied into the *pBuffer* parameter. In this instance, the *pszKey* parameter is ignored.

Note: If the enumeration cannot be performed for any reason, the default character string is *not* copied.

This function returns the length of the list, up to, but not including, the final null. If the enumerated key names exceed the available buffer space, the enumerated names are truncated, the enumerated list is not terminated with 2 bytes of zeros, and the *ulLength* parameter is set to the number of bytes copied into the *pBuffer* parameter.

This function is case-dependent; thus the strings in the *pszApp* parameter and the *pszKey* parameter must match exactly. This avoids any code-page dependency. The application storing the data must do any case-independent matching.

The enumeration call does not distinguish between data written with the [PrfWriteProfileString](#) function and the [PrfWriteProfileData](#) function.

PrfQueryProfileString - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_PARM (0x1303)

A parameter to the function contained invalid data.

PMERR_BUFFER_TOO_SMALL (0x110B)

The supplied buffer was not large enough for the data to be returned.

PMERR_NOT_IN_IDX (0x1304)

The application name, key-name or program handle was not found.

PMERR_CAN_NOT_CALL_SPOOLER (0x130D)

An error occurred attempting to call the spooler validation routine. This error is not raised if the spooler is not installed.

PMERR_INVALID_ASCII (0x130C)

The profile string is not a valid zero-terminated string.

PrfQueryProfileString - Related Functions

Related Functions

- [PrfWriteProfileString](#)

PrfQueryProfileString - Example Code

PrfQueryProfileString is issued twice to obtain the names of the default printer, the default presentation driver, and the queue associated with the printer. If any of these requests fails, the default values already defined in DEVOPENSTRUC are used.

```
#define INCL_WINSHELLDATA
#include <OS2.H>
char szTemp[80];
char szBuff[257];
```

```

PCH ptscan;

DEVOPENSTRUC dopPrinter = {"LPT1Q",
    (PSZ)"IBM4201",
    0L,
    (PSZ)"PM_Q_STD",
    0L, 0L, 0L, 0L, 0L};

if (PrfQueryProfileString(HINI_PROFILE,
    (PSZ)"PM_SPOOLER",
    (PSZ)"PRINTER",
    NULL,
    (PVOID)szTemp,
    (LONG)sizeof(szTemp)
    )){
    szTemp[strlen(szTemp)-1] = 0;
    if (PrfQueryProfileString(HINI_PROFILE,
        (PSZ)"PM_SPOOLER_PRINTER",
        (PSZ)szTemp,
        NULL,
        (PVOID)szBuff,
        (LONG)sizeof(szBuff)
        )){

        ptscan = (PCH)strchr(szBuff, ';');
        ptscan++;
        ptscan = (PCH)strchr(ptscan, (INT)';');
        ptscan++;
        *(ptscan + strcspn(ptscan, ".,;")) = 0;
        dopPrinter.pszLogAddress = ptscan;

        ptscan = (PCH)strchr(szBuff, (INT)';');
        ptscan++;
        *(ptscan + strcspn(ptscan, ".,;")) = 0;
        dopPrinter.pszDriverName = ptscan;

    }
}

```

PrfQueryProfileString - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

PrfReset

PrfReset - Syntax

This function defines which files are to be used as the user and system profiles.

```
#define INCL_WINSHELLDATA /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
PPRFPROFILE pPrfProfile; /* Profile-names structure. */
BOOL      rc;          /* Success indicator. */

rc = PrfReset(hab, pPrfProfile);
```

PrfReset Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

PrfReset Parameter - pPrfProfile

pPrfProfile ([PPRFPROFILE](#)) - input
Profile-names structure.

This contains the names of the files to be used as the new Presentation Manager* (PM*) profile files. Any valid file names can be used. A name that is not already fully qualified is taken to refer to the current directory.

If the user profile file does not exist, a new file is created.

The name of the system profile cannot be changed. It must be the name of the current system profile as returned by [PrfQueryProfile](#).

PrfReset Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

PrfReset - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

pPrfProfile ([PPRFPROFILE](#)) - input
Profile-names structure.

This contains the names of the files to be used as the new Presentation Manager* (PM*) profile files. Any valid file names can be used. A name that is not already fully qualified is taken to refer to the current directory.

If the user profile file does not exist, a new file is created.

The name of the system profile cannot be changed. It must be the name of the current system profile as returned by [PrfQueryProfile](#).

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

PrfReset - Remarks

This function causes the workstation to use different profiles. When the workstation is initialized, the names of the user and system profiles are taken from the PROTSHELL statement specified in CONFIG.SYS. PrfReset allows the profiles to be changed during operation of the workstation, for example by a logon application controlling multiple consecutive users of the system.

After the PrfReset function completes, the system has a new set of preferences (for example screen colors), a new start-up list, and new spooler parameters.

The PrfReset function broadcasts the [PL_ALTERED](#) message, which must be processed by all applications that read their default settings from the user or system profiles.

Note: This will only change the default system values in the ini file. It is up to the applications to read the new default settings and reset them to their new values.

For example, consider logon applications. On receipt of a [PL_ALTERED](#) message, they should carry out the following:

- Read the new color settings from the new profiles, and set the new screen colors (and palettes) which should be refreshed.
- Set the country information, for example the date and time format, which is read from the new profiles.
- Other preferences, for example, those that affect the operations of the alarm and the mouse, should also update with the new settings held in the new profiles.

This function requires the existence of a message queue.

PrfReset - Errors

Possible returns from [WinGetLastError](#)

PMERR_OPENING_INI_FILE (0x1301)
Unable to open initialization file (due to lack of disk space for example).

PrfReset - Related Functions

Related Functions

- [PrfQueryProfile](#)
-

PrfReset - Related Messages

Related Messages

- [PL_ALTERED](#)
-

PrfReset - Example Code

This function defines which files are to be used as the user and system profiles.

```
#define INCL_WINSHELLDATA
#include <OS2.H>
HAB hab;
char userpro[] = "profile.ini";
PRFPROFILE profile;

PrfQueryProfile(hab, &profile); /* get the system profile name */

profile.pszUserName = userpro;
profile.cchUserName = sizeof(userpro);

PrfReset (hab, &profile);
```

PrfReset - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

PrfWriteProfileData

PrfWriteProfileData - Syntax

This function writes a string of binary data into the specified profile.

```
#define INCL_WINSHELLDATA /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HINI    hini;        /* Initialization-file handle. */
PSZ     pszApp;      /* Application name. */
PSZ     pszKey;      /* Key name. */
PVOID   pData;       /* Value data. */
ULONG   ulDataLen;   /* Size of value data. */
BOOL    rc;          /* Success indicator. */

rc = PrfWriteProfileData(hini, pszApp, pszKey,
    pData, ulDataLen);
```

PrfWriteProfileData Parameter - hini

hini ([HINI](#)) - input

Initialization-file handle.

HINI_PROFILE

User profile

HINI_USERPROFILE

User profile

HINI_SYSTEMPROFILE

System profile

Other

Initialization-file handle returned by [PrfOpenProfile](#).

PrfWriteProfileData Parameter - pszApp

pszApp ([PSZ](#)) - input

Application name.

The case-dependent name of the application for which profile data is to be written. Names starting with the characters "PM_" are reserved for system use.

PrfWriteProfileData Parameter - pszKey

pszKey ([PSZ](#)) - input

Key name.

The case-dependent name of the key for which profile data is to be written.

This parameter can be NULL in which case *all* the *pszKey* or *pData* pairs associated with *pszApp* are deleted.

PrfWriteProfileData Parameter - pData

pData ([PVOID](#)) - input
Value data.

This is the value of the *pszKey* or *pData* pair that is written to the profile. It is *not* zero-terminated, and its length is given by the *ulDataLen* parameter.

If this parameter is NULL, the string associated with the *pszKey* parameter is deleted.

PrfWriteProfileData Parameter - ulDataLen

ulDataLen ([ULONG](#)) - input
Size of value data.

The size of the data to be written. This length should not exceed 64K bytes.

PrfWriteProfileData Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

PrfWriteProfileData - Parameters

hIni ([HINI](#)) - input
Initialization-file handle.

HINI_PROFILE	User profile
HINI_USERPROFILE	User profile
HINI_SYSTEMPROFILE	System profile
Other	Initialization-file handle returned by PrfOpenProfile .

pszApp ([PSZ](#)) - input
Application name.

The case-dependent name of the application for which profile data is to be written. Names starting with the characters "PM_" are reserved for system use.

pszKey ([PSZ](#)) - input
Key name.

The case-dependent name of the key for which profile data is to be written.

This parameter can be NULL in which case *all* the *pszKey* or *pData* pairs associated with *pszApp* are deleted.

pData ([PVOID](#)) - input
Value data.

This is the value of the *pszKey* or *pData* pair that is written to the profile. It is *not* zero-terminated, and its length is given by the *ulDataLen* parameter.

If this parameter is NULL, the data associated with the *pszKey* parameter is deleted.

ulDataLen ([ULONG](#)) - input
Size of value data.

The size of the data to be written. This length should not exceed 64K bytes.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

PrfWriteProfileData - Remarks

If there is no application field in the file that matches the *pszApp*, a new application field is created before the *pszKey* or *pszData* entry is made.

If the key name does not exist for the application, a new *pszKey* or *pszData* entry is created for that application. If the *pszKey* already exists in the file, the existing value is overwritten.

Because of the binary nature of the data, the input data is not zero-terminated. Thus, the length provided is the only way to identify the length of the data.

The maximum size of data that can be associated with a key name is 64K bytes. Data in excess of 64K bytes may be written, but there is no mechanism for retrieving all the data.

Deleting Profile Information

To remove the data associated with a given *pszKey* value for an application, specify a NULL value for *pData* and a zero length in *ulDataLen*.

To remove all the data associated with a particular application, use a *pszKey* value of NULL, a *pData* value of NULL, and a *ulDataLen* value of 0.

To remove **ALL** the data in a user profile, set *pszName*, *pszKey*, and *pData* to NULL, and set *ulDataLen* to zero.

PrfWriteProfileData - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_PARM (0x1303)
A parameter to the function contained invalid data.

PMERR_CAN_NOT_CALL_SPOOLER (0x130D)

An error occurred attempting to call the spooler validation routine. This error is not raised if the spooler is not installed.

PrfWriteProfileData - Related Functions

Related Functions

- [PrfQueryProfileSize](#)
-

PrfWriteProfileData - Example Code

This function writes, and subsequently reads, the profile data associated with application SAMPLE.EXE.

```
/* Write and read binary data to a profile.  Some return code */
/* checking omitted for brevity.                                */

#define INCL_WINSHELLDATA
#define INCL_WINERRORS
#define INCL_DOSERRORS
#include <os2.h>
#include <stdio.h>
#include <string.h>

#define BUFSIZE 7

INT main(VOID) {

    HAB    hab          = NULLHANDLE;          /* Window handle */
    HINI    hini         = NULLHANDLE;          /* INI file handle */
    PSZ     pszFileName  = "PROFILE.INI";        /* Name of profile */
    BOOL    rc           = FALSE;               /* Return code */
    PSZ     pszAppName   = "SAMPLE.EXE";        /* Application name */
    PSZ     pszKeyName   = "BINARY_DATA";       /* Data key name */
    APIRET  ret          = NO_ERROR;            /* API return code */
    PBYTE   pData        = NULL;               /* Pointer to data buffer */
    ULONG   ulDataSize   = 0;                  /* Size of data to read */
    INT     idx          = 0;                  /* Loop index */

    hab = WinInitialize( 0 );
    hini = PrfOpenProfile( hab, pszFileName ); /* Open INI file */

    /* Allocate Memory for binary data to be written and read */

    ret = DosAllocMem( (PPVOID)&pData, sizeof(BYTE)*80,
                      (ULONG)PAG_COMMIT | PAG_READ | PAG_WRITE );

    /* Initialize binary data and then write it to profile */

    for(idx = 0; idx < BUFSIZE; idx++) {
        pData[idx] = idx+65;
    }

    rc = PrfWriteProfileData( hini, pszAppName, pszKeyName,
                             (PVOID)pData, (ULONG)BUFSIZE );

    if(rc == FALSE) {
        printf("PrfWriteProfileData error code: %X\n", WinGetLastError(hab));
        return 1;
    }

    /* Put junk in buffer so we can see if read is successful */

    for(idx = 0; idx < BUFSIZE; idx++) {
        pData[idx] = idx;
    }
}
```

```

        /* Retrieve size of data and then get data */

rc = PrfQueryProfileSize( hini, pszAppName, pszKeyName, &ulDataSize );
rc = PrfQueryProfileData( hini, pszAppName, pszKeyName,
                          (PVOID)pData, &ulDataSize);

if(rc==FALSE) {
    printf("PrfQueryProfileData error code: %X\n", WinGetLastError(hab));
    return 1;
}

printf("Profile Data Read:");
for(idx=0; idx < ulDataSize; idx++)
    printf("%c", pData[idx]);
printf("\n");

PrfCloseProfile(hini);      /* Close profile */
DosFreeMem(pData);         /* Free memory */
return NO_ERROR;           /* Phone home <g> */
}

```

PrfWriteProfileData - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

PrfWriteProfileString

PrfWriteProfileString - Syntax

This function writes a string of character data into the specified profile.

```

#define INCL_WINSHELLDATA /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HINI    hini;      /* Initialization-file handle. */
PSZ     pszApp;    /* Application name. */
PSZ     pszKey;    /* Key name. */
PSZ     pszData;   /* Text string. */
BOOL    rc;        /* Success indicator. */

rc = PrfWriteProfileString(hini, pszApp, pszKey,
                           pszData);

```

PrfWriteProfileString Parameter - hini

hini ([HINI](#)) - input

Initialization-file handle.

HINI_PROFILE

User profile

HINI_USERPROFILE

User profile

HINI_SYSTEMPROFILE

System profile

Other

Initialization-file handle returned by [PrfOpenProfile](#).

PrfWriteProfileString Parameter - pszApp

pszApp ([PSZ](#)) - input

Application name.

The case-dependent name of the application for which profile data is to be written. Names starting with the characters "PM_" are reserved for system use.

PrfWriteProfileString Parameter - pszKey

pszKey ([PSZ](#)) - input

Key name.

The case-dependent name of the key for which profile data is to be written.

This parameter can be NULL, in which case *all* the *pszKey* or *pszData* pairs associated with the *pszApp* parameter are deleted.

PrfWriteProfileString Parameter - pszData

pszData ([PSZ](#)) - input

Text string.

This is the value of the *pszKey* or *pszData* pair that is written to the profile.

If this parameter is NULL, the string associated with the *pszKey* is deleted (that is, the entry is deleted).

If this parameter is not NULL, the string is used as the value of the *pszKey* or *pszData* pair, even if the string has zero length.

PrfWriteProfileString Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

PrfWriteProfileString - Parameters

hini ([HINI](#)) - input
Initialization-file handle.

HINI_PROFILE	User profile
HINI_USERPROFILE	User profile
HINI_SYSTEMPROFILE	System profile
Other	Initialization-file handle returned by PrfOpenProfile .

pszApp ([PSZ](#)) - input
Application name.

The case-dependent name of the application for which profile data is to be written. Names starting with the characters "PM_" are reserved for system use.

pszKey ([PSZ](#)) - input
Key name.

The case-dependent name of the key for which profile data is to be written.

This parameter can be NULL, in which case *all* the *pszKey* or *pszData* pairs associated with the *pszApp* parameter are deleted.

pszData ([PSZ](#)) - input
Text string.

This is the value of the *pszKey* or *pszData* pair that is written to the profile.

If this parameter is NULL, the string associated with the *pszKey* is deleted (that is, the entry is deleted).

If this parameter is not NULL, the string is used as the value of the *pszKey* or *pszData* pair, even if the string has zero length.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

PrfWriteProfileString - Remarks

If there is no application field in the file that matches the *pszApp*, a new application field is created before the *pszKey* or *pszData* entry is made.

If the key name does not exist for the application, a new *pszKey* or *pszData* entry is created for that application. If the *pszKey* already exists in the file, the existing value is overwritten.

PrfWriteProfileString - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_PARM (0x1303)

A parameter to the function contained invalid data.

PMERR_CAN_NOT_CALL_SPOOLER (0x130D)

An error occurred attempting to call the spooler validation routine. This error is not raised if the spooler is not installed.

PrfWriteProfileString - Related Functions

Related Functions

- [PrfQueryProfileString](#)

PrfWriteProfileString - Example Code

This function writes and then reads a string written to the profile associated with the application STRINGY.EXE.

```
/* This programs writes and then read string data to a profile. */
/* Some error checking has been omitted for brevity. */

#define INCL_WINSHELLDATA
#define INCL_WINERRORS
#define INCL_DOSERRORS
#include <os2.h>
#include <stdio.h>
#include <string.h>

INT main(VOID) {

    HAB    hab          = NULLHANDLE;
    HINI    hini         = NULLHANDLE;
    PSZ     pszFileName  = "STRINGY.INI";
    BOOL     rc          = TRUE;
    ULONG    i           = 0L;
    PSZ     pszAppName   = "STRINGY.EXE";
    PSZ     pszKeyName   = "STATUS";
    PSZ     pszString    = "The string is alive.";
    PSZ     pszStringRead = NULL;
    APIRET   ret         = NO_ERROR;
    ULONG    ulDataSize  = 0L;

    hab = WinInitialize( 0 );

    hini = PrfOpenProfile( hab, pszFileName );

    rc = PrfWriteProfileString( hini, pszAppName, pszKeyName, pszString);
```

```

if(rc == FALSE) {
    printf("PrfWriteProfileString error code: %X\n", WinGetLastError(hab));
    return 1;
}

/* Retrieve size of string in profile, and allocate memory for it */

rc = PrfQueryProfileSize( hini, pszAppName, pszKeyName, &ulDataSize );
ret = DosAllocMem( (PVOID)&pszStringRead, ulDataSize,
    (ULONG)PAG_COMMIT | PAG_READ | PAG_WRITE );

/* Retrieve data string */

rc = PrfQueryProfileString( hini, pszAppName, pszKeyName,
    (PSZ)"Error Retrieving Data",
    (PVOID) pszStringRead,
    ulDataSize );

if(rc==FALSE) {
    printf("PrfQueryProfileData error code: %X\n", WinGetLastError(hab));
    return 1;
}

printf("Profile String read: '%s'\n", pszStringRead );

PrfCloseProfile( hini );          /* Close the profile      */
DosFreeMem( pszStringRead );      /* Free memory         */
return NO_ERROR;                  /* Return to the caller */
}

```

PrfWriteProfileString - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

Spooler Functions

This section describes functions that an application would use to write data directly to a spool file.

SplControlDevice

SplControlDevice - Syntax

This function cancels, holds, continues, or restarts a print device.

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where print device is to be controlled. */
PSZ      pszPortName;     /* Port name. */
ULONG    ulControl;        /* Operation to perform. */
SPLERR   rc;               /* Return code. */

rc = SplControlDevice(pszComputerName, pszPortName,
                     ulControl);
```

SplControlDevice Parameter - pszComputerName

pszComputerName ([PSZ](#)) - input
Name of computer where print device is to be controlled.

A NULL string specifies the local workstation.

SplControlDevice Parameter - pszPortName

pszPortName ([PSZ](#)) - input
Port name.

SplControlDevice Parameter - ulControl

ulControl ([ULONG](#)) - input
Operation to perform.

PRD_DELETE	Delete current print job
PRD_PAUSE	Pause printing
PRD_CONT	Continue paused print job
PRD_RESTART	Restart print job.

SplControlDevice Return Value - rc

rc (**SPLERR**) - returns

Return code.

NO_ERROR (0)

No errors occurred.

ERROR_NOT_SUPPORTED (50)

This request is not supported by the network.

ERROR_BAD_NETPATH (53)

The network path cannot be located.

NERR_NetNotStarted (2102)

The network program is not started.

NERR_DestNotFound (2152)

The print device cannot be found.

NERR_DestIdle (2158)

This print device is idle and cannot accept control operations.

NERR_DestInvalidOp (2159)

This print device request contains an invalid control function.

NERR_ProcNoRespond (2160)

The queue processor is not responding.

NERR_SpoolerNotLoaded (2161)

The spooler is not running.

NERR_InvalidComputer (2351)

The computer name is invalid.

SplControlDevice - Parameters

pszComputerName (**PSZ**) - input

Name of computer where print device is to be controlled.

A NULL string specifies the local workstation.

pszPortName (**PSZ**) - input

Port name.

ulControl (**ULONG**) - input

Operation to perform.

PRD_DELETE

Delete current print job

PRD_PAUSE

Pause printing

PRD_CONT

Continue paused print job

PRD_RESTART

Restart print job.

rc (**SPLERR**) - returns

Return code.

NO_ERROR (0)

No errors occurred.

ERROR_NOT_SUPPORTED (50)

This request is not supported by the network.

ERROR_BAD_NETPATH (53)

The network path cannot be located.

NERR_NetNotStarted (2102)

The network program is not started.

NERR_DestNotFound (2152)

The print device cannot be found.

NERR_DestIdle (2158)

This print device is idle and cannot accept control operations.

NERR_DestInvalidOp (2159)

This print device request contains an invalid control function.

NERR_ProcNoRespond (2160)

The queue processor is not responding.

NERR_SpoolerNotLoaded (2161)
The spooler is not running.
NERR_InvalidComputer (2351)
The computer name is invalid.

SplControlDevice - Remarks

A paused print device cannot accept new print jobs.

If PRD_DELETE is attempted when there is no current print job, NERR_DestIdle (2158) is returned.

To control jobs on a remote server requires administrator privilege.

SplControlDevice - Related Functions

Related Functions

- [SplEnumDevice](#)
 - [SplQueryDevice](#)
-

SplControlDevice - Example Code

This sample code demonstrates the result of various actions that can be performed on the print device by this function call. At the command line, a print device name is entered along with an action code.

```
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>          /* for printf function */

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    SPLERR splerr ;
    ULONG  ulControl=0L ;
    PSZ     pszComputerName = NULL ;
    PSZ     pszPrintDeviceName ;

    /* Input a Print Device Name and an Action Code on the command line */
    if (argc != 3)
    {
        printf("Syntax is: qcontrol PrintDeviceName ActionCode \n");
        printf("Action codes are: D-Delete, P-Pause, C-Continue, R-Restart\n\n");
        DosExit( EXIT_PROCESS , 0 ) ;
    }

    /* Get the print device name from the first input parameter. */
    pszPrintDeviceName = argv[1];

    /* Get the action code from the second input parameter. */
    switch (argv[2][0])
    {
        case 'D':
            ulControl = PRD_DELETE ;
            break;
    }
```

```

        case 'P':
            ulControl = PRD_PAUSE ;
            break;
        case 'C':
            ulControl = PRD_CONT ;
            break;
        case 'R':
            ulControl = PRD_RESTART ;
            break;
        default:
            printf("Invalid code\n");
            DosExit( EXIT_PROCESS , 0 ) ;
    }
    /* Call the function with the parameters obtained from the command line. */
    splerr = SplControlDevice(pszComputerName, pszPrintDeviceName, ulControl);

    /* If there is an error returned, print it. */
    if (splerr != 0L)
    {
        switch (splerr)
        {
            case NERR_DestNotFound :
                printf("Destination does not exist.\n");
                break;
            case NERR_DestIdle:
                printf("This print device is idle - can't do control ops. \n");
                break;
            default:
                printf("Errorcode = %ld\n",splerr);
        }
    } else {
        printf("The print job operation was performed.\n\n");
    }
    DosExit( EXIT_PROCESS , 0 ) ;
    return (splerr) ;
}

```

SplControlDevice - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

SplCopyJob

SplCopyJob - Syntax

This function copies a job in a print queue.

Currently there is a restriction that a job can only be copied onto the same queue (and computer) as the original job.

```

#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      pszSrcComputerName; /* Name of computer where job is to be copied from. */
PSZ      pszSrcQueueName;   /* Name of queue where job is to be copied from. */
ULONG    ulSrcJob;          /* Source Job identification number. */
PSZ      pszTrgComputerName; /* Name of computer where job is to be copied to. */
PSZ      pszTrgQueueName;   /* Name of queue where job is to be copied to. */
PULONG   pulTrgJob;         /* Job identification number of new job. */
SPLERR    rc;               /* Return code. */

rc = SplCopyJob(pszSrcComputerName, pszSrcQueueName,
               ulSrcJob, pszTrgComputerName, pszTrgQueueName,
               pulTrgJob);

```

SplCopyJob Parameter - pszSrcComputerName

pszSrcComputerName (**PSZ**) - input
 Name of computer where job is to be copied from.

A NULL string specifies the local workstation.

SplCopyJob Parameter - pszSrcQueueName

pszSrcQueueName (**PSZ**) - input
 Name of queue where job is to be copied from.

SplCopyJob Parameter - ulSrcJob

ulSrcJob (**ULONG**) - input
 Source Job identification number.

SplCopyJob Parameter - pszTrgComputerName

pszTrgComputerName (**PSZ**) - input
 Name of computer where job is to be copied to.

A NULL string specifies the local workstation.

SplCopyJob Parameter - pszTrgQueueName

pszTrgQueueName (PSZ) - input

Name of queue where job is to be copied to.

A NULL string specifies the same queue as the original job.

SplCopyJob Parameter - pulTrgJob

pulTrgJob (PULONG) - output

Job identification number of new job.

SplCopyJob Return Value - rc

rc (SPLERR) - returns

Return code.

NO_ERROR (0)

No errors occurred.

ERROR_ACCESS_DENIED (5)

Access is denied.

ERROR_NOT_SUPPORTED (50)

This request is not supported by the network.

ERROR_INVALID_PARAMETER (87)

An invalid parameter is specified.

NERR_NetNotStarted (2102)

The network program is not started.

NERR_QNotFound (2150)

The printer queue does not exist.

NERR_JobNotFound (2151)

The print job does not exist.

NERR_SpoolerNotLoaded (2161)

The spooler is not running.

NERR_InvalidComputer (2351)

The computer name is invalid.

SplCopyJob - Parameters

pszSrcComputerName (PSZ) - input

Name of computer where job is to be copied from.

A NULL string specifies the local workstation.

pszSrcQueueName (PSZ) - input

Name of queue where job is to be copied from.

ulSrcJob (ULONG) - input

Source Job identification number.

pszTrgComputerName ([PSZ](#)) - input
Name of computer where job is to be copied to.

A NULL string specifies the local workstation.

pszTrgQueueName ([PSZ](#)) - input
Name of queue where job is to be copied to.

A NULL string specifies the same queue as the original job.

pulTrgJob ([PULONG](#)) - output
Job identification number of new job.

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)
No errors occurred.
ERROR_ACCESS_DENIED (5)
Access is denied.
ERROR_NOT_SUPPORTED (50)
This request is not supported by the network.
ERROR_INVALID_PARAMETER (87)
An invalid parameter is specified.
NERR_NetNotStarted (2102)
The network program is not started.
NERR_QNotFound (2150)
The printer queue does not exist.
NERR_JobNotFound (2151)
The print job does not exist.
NERR_SpoolerNotLoaded (2161)
The spooler is not running.
NERR_InvalidComputer (2351)
The computer name is invalid.

SplCopyJob - Related Functions

Related Functions

- [SplEnumJob](#)
 - [SplEnumQueue](#)
 - [SplQueryJob](#)
 - [SplQueryQueue](#)
-

SplCopyJob - Example Code

This sample code will make a duplicate copy of the jobid that is entered at the prompt. Presently, there is a restriction that the job can only be duplicated on the same computer/queue; i.e. a local job.

```
#define INCL_SPL
#include <os2.h>
#include <stdio.h>      /* for printf function */
#include <stdlib.h>     /* for atoi function  */

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    SPLERR splerr ;
```

```

ULONG   ulSrcJob, ulTrgJob ;
PSZ      pszSrcComputerName, pszTrgComputerName ;
PSZ      pszSrcQueueName, pszTrgQueueName ;

if (argc != 2)
{
    printf("Command is: copyjob JOBID\n");
    DosExit( EXIT_PROCESS , 0 ) ;
}
pszSrcComputerName = (PSZ)NULL ;

/* The only valid values at present for these three parameters is NULL */
pszSrcQueueName = (PSZ)NULL;
pszTrgComputerName = (PSZ)NULL ;
pszTrgQueueName = (PSZ)NULL ;

/* Convert input parameter to a ULONG */
ulSrcJob = atoi ( argv[1] );

if (splerr = SplCopyJob(pszSrcComputerName, pszSrcQueueName, ulSrcJob,
                        pszTrgComputerName, pszTrgQueueName, &ulTrgJob))
{
    printf("Return code SplCopyJob = %d\n", splerr);
}
else
{
    printf("New job ID is %d\n", ulTrgJob);
}
DosExit( EXIT_PROCESS , 0 ) ;
return (splerr);
} /* end main */

```

SplCopyJob - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

SplCreateDevice

SplCreateDevice - Syntax

This function establishes a print device on the local workstation or a remote server.

```

#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where print device is to be added. */
ULONG    ulLevel;         /* Level of detail provided. */
PVOID     pBuf;           /* Data structure. */

```

```
ULONG      cbBuf;          /* Size, in bytes, of data structure. */
SPLERR     rc;             /* Return code. */

rc = SplCreateDevice(pszComputerName, ulLevel,
                    pBuf, cbBuf);
```

SplCreateDevice Parameter - pszComputerName

pszComputerName ([PSZ](#)) - input
Name of computer where print device is to be added.

A NULL string specifies the local workstation.

SplCreateDevice Parameter - ulLevel

ulLevel ([ULONG](#)) - input
Level of detail provided.

This must be 3.

SplCreateDevice Parameter - pBuf

pBuf ([PVOID](#)) - input
Data structure.

It points to a [PRDINFO3](#) data structure.

SplCreateDevice Parameter - cbBuf

cbBuf ([ULONG](#)) - input
Size, in bytes, of data structure.

SplCreateDevice Return Value - rc

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)
No errors occurred.

ERROR_ACCESS_DENIED (5)
Access is denied.

ERROR_NOT_SUPPORTED (50)
This request is not supported by the network.

ERROR_BAD_NETPATH (53)
The network path cannot be located.

ERROR_INVALID_PARAMETER (87)
An invalid parameter is specified.

ERROR_INVALID_NAME (123)
The computer name is invalid.

ERROR_INVALID_LEVEL (124)
The level parameter is invalid.

NERR_NetNotStarted (2102)
The network program is not started.

NERR_BufTooSmall (2123)
The API return buffer is too small.

NERR_DestExists (2153)
The print device already exists.

NERR_DestNoRoom (2157)
The maximum number of print devices has been reached.

NERR_DestInvalidState (2162)
This operation cannot be performed on the print device.

NERR_SpoolNoMemory (2165)
A spooler memory allocation failure occurred.

NERR_DriverNotFound (2166)
The device driver does not exist.

NERR_BadDev (2341)
The device is already in use as a communications device.

NERR_InvalidComputer (2351)
The computer name is invalid.

SplCreateDevice - Parameters

pszComputerName ([PSZ](#)) - input
Name of computer where print device is to be added.

A NULL string specifies the local workstation.

ulLevel ([ULONG](#)) - input
Level of detail provided.

This must be 3.

pBuf ([PVOID](#)) - input
Data structure.

It points to a [PRDINFO3](#) data structure.

cbBuf ([ULONG](#)) - input
Size, in bytes, of data structure.

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)
No errors occurred.

ERROR_ACCESS_DENIED (5)
Access is denied.

ERROR_NOT_SUPPORTED (50)
This request is not supported by the network.

ERROR_BAD_NETPATH (53)
The network path cannot be located.

ERROR_INVALID_PARAMETER (87)
An invalid parameter is specified.

ERROR_INVALID_NAME (123)
The computer name is invalid.

ERROR_INVALID_LEVEL (124)
The level parameter is invalid.

NERR_NetNotStarted (2102)
The network program is not started.

NERR_BufTooSmall (2123)
The API return buffer is too small.

NERR_DestExists (2153)
The print device already exists.

NERR_DestNoRoom (2157)
The maximum number of print devices has been reached.

NERR_DestInvalidState (2162)
This operation cannot be performed on the print device.

NERR_SpoolNoMemory (2165)
A spooler memory allocation failure occurred.

NERR_DriverNotFound (2166)
The device driver does not exist.

NERR_BadDev (2341)
The device is already in use as a communications device.

NERR_InvalidComputer (2351)
The computer name is invalid.

SplCreateDevice - Remarks

The result of this function is the creation of a new print device definition.

The printer is set up to print on the logical address (port) defined by in [PRDINFO3](#). If is NULL, the print device definition is created but is not connected to any logical address. In this case no printing can occur on that print device or from any print queue connected only to that print device. If a logical address is specified, it must already be defined in the PM_SPOOLER_PORTS section of the initialization file.

Note: To change the connection between a print device and a port, use [SplSetDevice](#).

All device drivers and queues specified with the print device must already be defined to the spooler.

To add a remote print device requires administrator privilege.

SplCreateDevice - Related Functions

Related Functions

- [SplDeleteDevice](#)
 - [SplEnumDevice](#)
 - [SplEnumDriver](#)
 - [SplEnumPort](#)
-

SplCreateDevice - Example Code

This sample code creates a [PRDINFO3](#) structure with dummy parameters. This structure is then used to call SplCreateDevice to establish a print device on a local workstation.

```
#define INCL_BASE
```

```

#define INCL_DOSMEMMGR
#define INCL_SPL
#define INCL_SPLDOSPRINT

#include <os2.h>
#include <stdio.h>      /* for printf function */
#include <string.h>     /* for strcpy function */

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    ULONG    splerr    ;
    ULONG    cbBuf;
    ULONG    ulLevel  ;
    PSZ      pszComputerName ;
    PSZ      pszPrintDeviceName ;
    PRDINFO3 prd3    ;

    if (argc != 2)
    {
        printf("Syntax:  sdprt  DeviceName \n");
        DosExit( EXIT_PROCESS , 0 ) ;
    }
    /* We are going to create a print device on the local workstation. */
    pszComputerName = (PSZ)NULL ;

    /* Get the name from the command line. */
    pszPrintDeviceName = argv[1];

    /* Level 3 is valid. We will use level 3. */
    ulLevel = 3;

    /* Get size of buffer needed for a PRDINFO3 structure. */
    cbBuf = sizeof(PRDINFO3);

    /* Set up the structure with dummy parameters. */
    prd3.pszPrinterName = pszPrintDeviceName;
    prd3.pszUserName = NULL;
    prd3.pszLogAddr = "LPT1";
    prd3.uJobId=0;
    prd3.pszComment= "Test comment";
    prd3.pszDrivers = "IBMNULL";
    prd3.usTimeOut = 777;

    /* Make the call. */
    splerr = SplCreateDevice(pszComputerName, ulLevel,
                            &prd3, cbBuf);

    /* Print out the results. */
    if (splerr == NO_ERROR)
        printf("The device was successfully created.");
    else
        printf("SplCreateDevice Error=%ld, cbNeeded=%ld\n",
               splerr, cbBuf) ;

    DosExit( EXIT_PROCESS , 0 ) ;
    return (splerr);
}

```

SplCreateDevice - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

SplCreatePort

SplCreatePort - Syntax

Description:

This function is called to create a printer port. The printer port can be a virtual port.

Note: There is currently no support for using this API to remotely add a printer port.

```
#define INCL_SPL
#define INCL_SPLBIDI
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where port is to be installed. */
PSZ      pszPortName;     /* Name of printer port to install. */
PSZ      pszPortDriver;   /* Name of the port driver assigned to this port. */
ULONG    ulVersion;       /* Version of port to create. */
PVOID    pBuf;            /* Port-driver-specific buffer. */
ULONG    cbBuf;           /* Length of the data in pBuf, in bytes. */
ULONG    rc;              /* Return codes. */

rc = SplCreatePort(pszComputerName, pszPortName,
                  pszPortDriver, ulVersion, pBuf, cbBuf);
```

SplCreatePort Parameter - pszComputerName

pszComputerName ([PSZ](#)) - input

Name of computer where port is to be installed.

A NULL string is used for installing a port on the local machine. This parameter currently must be NULL.

SplCreatePort Parameter - pszPortName

pszPortName ([PSZ](#)) - input

Name of printer port to install.

This port, virtual or regular, must not already exist. Following is an example of a port name:
LPT5

SplCreatePort Parameter - pszPortDriver

pszPortDriver ([PSZ](#)) - input

Name of the port driver assigned to this port.

If not NULL, this field must specify an installed port driver; it is case-sensitive. Following is an example of a port driver:
PARALLEL

SplCreatePort Parameter - ulVersion

ulVersion ([ULONG](#)) - input

Version of port to create.

Values are as follows:

SPLPORT_VERSION_REGULAR - 0

Creates normal printer port that can be connected to a print queue. This port will remain after a reboot.

SPLPORT_VERSION_VIRTUAL - 1

Creates a virtual printer port that can only be used for querying and setting. This port exists only until the spooler is disabled or the machine is rebooted.

SplCreatePort Parameter - pBuf

pBuf ([PVOID](#)) - input

Port-driver-specific buffer.

If SPLPORT_VERSION_VIRTUAL is set, this is a port-driver-specific buffer that contains information necessary for the port driver to set up a virtual connection to the printer. There cannot be any imbedded pointers in this buffer. This buffer will be passed to the port driver with the BIDI_ADD_VIRTUAL_PORT command.

SplCreatePort Parameter - cbBuf

cbBuf ([ULONG](#)) - input

Length of the data in *pBuf*, in bytes.

SplCreatePort Return Value - rc

rc (**ULONG**) - returns
Return codes.

0	Success
ERROR_DEVICE_IN_USE(99)	Port name is already defined.
ERROR_FILE_NOT_FOUND(2)	Port driver name is not valid.
ERROR_INVALID_LEVEL(124)	<i>ulVersion</i> is not 0 or 1.
ERROR_INVALID_PARAMETER(87)	An invalid buffer was given.
ERROR_NOT_ENOUGH_MEMORY(8)	Not enough memory to satisfy request.
ERROR_NOT_SUPPORTED(50)	<i>pszComputerName</i> is not supported, or the port driver does not support the virtual port.
PMERR_SPL_SPOOLER_NOT_INSTALLED(0x4009)	Spooler not enabled; virtual ports require spooler to be enabled.

SplCreatePort - Parameters

pszComputerName (**PSZ**) - input
Name of computer where port is to be installed.

A NULL string is used for installing a port on the local machine. This parameter currently must be NULL.

pszPortName (**PSZ**) - input
Name of printer port to install.

This port, virtual or regular, must not already exist. Following is an example of a port name:
LPT5

pszPortDriver (**PSZ**) - input
Name of the port driver assigned to this port.

If not NULL, this field must specify an installed port driver; it is case-sensitive. Following is an example of a port driver:
PARALLEL

ulVersion (**ULONG**) - input
Version of port to create.

Values are as follows:

SPLPORT_VERSION_REGULAR - 0
Creates normal printer port that can be connected to a print queue. This port will remain after a reboot.

SPLPORT_VERSION_VIRTUAL - 1
Creates a virtual printer port that can only be used for querying and setting. This port exists only until the spooler is disabled or the machine is rebooted.

pBuf (**PVOID**) - input

Port-driver-specific buffer.

If SPLPORT_VERSION_VIRTUAL is set, this is a port-driver-specific buffer that contains information necessary for the port driver to set up a virtual connection to the printer. There cannot be any imbedded pointers in this buffer. This buffer will be passed to the port driver with the BIDI_ADD_VIRTUAL_PORT command.

cbBuf ([ULONG](#)) - input
Length of the data in *pBuf*, in bytes.

rc ([ULONG](#)) - returns
Return codes.

0	Success
ERROR_DEVICE_IN_USE(99)	Port name is already defined.
ERROR_FILE_NOT_FOUND(2)	Port driver name is not valid.
ERROR_INVALID_LEVEL(124)	<i>ulVersion</i> is not 0 or 1.
ERROR_INVALID_PARAMETER(87)	An invalid buffer was given.
ERROR_NOT_ENOUGH_MEMORY(8)	Not enough memory to satisfy request.
ERROR_NOT_SUPPORTED(50)	<i>pszComputerName</i> is not supported, or the port driver does not support the virtual port.
PMERR_SPL_SPOOLER_NOT_INSTALLED(0x4009)	Spooler not enabled; virtual ports require spooler to be enabled.

SplCreatePort - Remarks

None.

SplCreatePort - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Glossary](#)
-

SplCreateQueue

SplCreateQueue - Syntax

This function creates a new print queue on the local workstation or on a remote server. A remote server setup requires the LAN Requester and Server software.

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where queue is to be created. */
ULONG    ulLevel;         /* Level of detail provided. */
PVOID     pbBuf;          /* Data structure. */
ULONG     cbBuf;          /* Size, in bytes, of data structure. */
SPLERR    rc;             /* Return code. */

rc = SplCreateQueue(pszComputerName, ulLevel,
                   pbBuf, cbBuf);
```

SplCreateQueue Parameter - pszComputerName

pszComputerName (**PSZ**) - input
Name of computer where queue is to be created.

A NULL string specifies a local workstation.

SplCreateQueue Parameter - ulLevel

ulLevel (**ULONG**) - input
Level of detail provided.

This must be 3 or 6.

SplCreateQueue Parameter - pbBuf

pbBuf (**PVOID**) - input
Data structure.

It points to a data structure depending on the value specified in *ulLevel* as follows:

	<i>ulLevel</i>	Buffer Content.
3		a PRQINFO3 data structure
6		a PRQINFO6 data structure

SplCreateQueue Parameter - cbBuf

cbBuf ([ULONG](#)) - input
Size, in bytes, of data structure.

It must be greater than 0.

SplCreateQueue Return Value - rc

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)
No errors occurred.

ERROR_NOT_SUPPORTED (50)
This request is not supported by the network.

ERROR_INVALID_PARAMETER (87)
An invalid parameter is specified.

ERROR_INVALID_NAME (123)
The computer name is invalid.

ERROR_INVALID_LEVEL (124)
The level parameter is invalid.

NERR_NetNotStarted (2102)
The network program is not started.

NERR_RedirectedPath (2117)
The operation is invalid on a redirected resource.

NERR_BufTooSmall (2123)
The API return buffer is too small.

NERR_DestNotFound (2152)
The printer destination cannot be found.

NERR_QExists (2154)
The printer queue already exists.

NERR_DestInvalidState (2162)
This operation cannot be performed on the print destination in its current state.

NERR_SpoolNoMemory (2165)
A spooler memory allocation failure occurred.

NERR_DriverNotFound (2166)
The device driver does not exist.

NERR_DataTypeInvalid (2167)
The data type is not supported by the queue processor.

NERR_ProcNotFound (2168)
The queue processor is not installed.

NERR_BadDev (2341)
The requested device is invalid.

NERR_CommDevInUse (2343)
This device is already in use as a communications device.

NERR_InvalidComputer (2351)
The computer name is invalid.

SplCreateQueue - Parameters

pszComputerName ([PSZ](#)) - input
Name of computer where queue is to be created.

A NULL string specifies a local workstation.

ulLevel ([ULONG](#)) - input

Level of detail provided.

This must be 3 or 6.

pbBuf ([PVOID](#)) - input
Data structure.

It points to a data structure depending on the value specified in *uLevel* as follows:

	<i>uLevel</i>	Buffer Content.
3		a PRQINFO3 data structure
6		a PRQINFO6 data structure

cbBuf ([ULONG](#)) - input
Size, in bytes, of data structure.

It must be greater than 0.

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_NAME (123)	The computer name is invalid.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_RedirectedPath (2117)	The operation is invalid on a redirected resource.
NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_DestNotFound (2152)	The printer destination cannot be found.
NERR_QExists (2154)	The printer queue already exists.
NERR_DestInvalidState (2162)	This operation cannot be performed on the print destination in its current state.
NERR_SpoolNoMemory (2165)	A spooler memory allocation failure occurred.
NERR_DriverNotFound (2166)	The device driver does not exist.
NERR_DataTypeInvalid (2167)	The data type is not supported by the queue processor.
NERR_ProcNotFound (2168)	The queue processor is not installed.
NERR_BadDev (2341)	The requested device is invalid.
NERR_CommDevInUse (2343)	This device is already in use as a communications device.
NERR_InvalidComputer (2351)	The computer name is invalid.

SplCreateQueue - Remarks

To create a queue on a remote server requires administrator privilege. The following fields are required in [PRQINFO3](#) or [PRQINFO6](#):

- *uPriority*
- *uStartTime*
- *uUntilTime*
- *pszSepFile*

- *pszParams*

If a queue of the name specified in *pszName* already exists on *pszComputerName*, the call fails unless the queue is marked for deletion. In this case, the queue is not deleted, and the creation fields are used to perform a [SplSetQueue](#) function on the queue.

If *pszPrinters* is NULL, the queue is created but not connected to any printer.

pszDriverName can be a NULL string, in which case *pDriverData* is ignored. Otherwise, *pszDriverName* must refer to the name of a device driver that is already defined in the initialization file (for example, "IBM4019").

SplCreateQueue - Related Functions

Related Functions

- [SplDeleteQueue](#)
- [SplEnumDevice](#)
- [SplEnumDriver](#)
- [SplEnumQueueProcessor](#)

SplCreateQueue - Example Code

This sample code creates a queue on the local workstation. The queue is created with dummy parameters. The name is entered at the command line.

```
#define INCL_BASE
#define INCL_SPL
#define INCL_SPLDOSPRINT

#include <os2.h>
#include <stdio.h>
#include <string.h>

INT main (argc, argv )
{
    INT argc;
    CHAR *argv[];

    ULONG splerr ;
    ULONG cbBuf;
    ULONG ulLevel ;
    PSZ pszComputerName ;
    PSZ pszQueueName ;
    PRQINFO3 prq3 ;

    if (argc != 2)
    {
        printf("Syntax:  sqcrt QueueName \n");
        DosExit( EXIT_PROCESS , 0 ) ;
    }

    pszComputerName = (PSZ)NULL ;
    ulLevel = 3L;

    /* Get the queue name from the argument entered at */
    /* the command line. */
    pszQueueName = argv[1];

    /* Determine the size of the needed buffer. */
    cbBuf = sizeof(PRQINFO3);

    /* Set up the structure with some dummy parameters. */
    prq3.pszName = pszQueueName;
    prq3.uPriority=5;
    prq3.uStartTime=0;
```

```

prq3.uUntilTime=0;
prq3.pszSepFile="c:\\os2\\sample.sep";
prq3.pszParms=NULL;
prq3.pszPrinters=NULL;
prq3.pszDriverName=NULL;
prq3.pDriverData="IBMNULL";      /* Set to Driver.Device name */

/* Make the call with the proper parameters. */
splerr = SplCreateQueue(pszComputerName, ulLevel,
                        &prq3, cbBuf);

/* Print out the error return code and some other information. */
printf("SplCreateQueue Error=%ld, cbNeeded=%ld\n",
       splerr, cbBuf) ;

DosExit( EXIT_PROCESS , 0 ) ;
return (splerr);
}

```

SplCreateQueue - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

SplDeleteDevice

SplDeleteDevice - Syntax

This function deletes a print device.

```

#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      pszComputerName;    /* Name of computer where print device is. to be deleted. */
PSZ      pszPrintDeviceName; /* Name of Print Device. */
SPLERR   rc;                 /* Return code. */

rc = SplDeleteDevice(pszComputerName, pszPrintDeviceName);

```

SplDeleteDevice Parameter - pszComputerName

pszComputerName (PSZ) - input
Name of computer where print device is. to be deleted.

A NULL string specifies the local workstation.

SplDeleteDevice Parameter - pszPrintDeviceName

pszPrintDeviceName (PSZ) - input
Name of Print Device.

SplDeleteDevice Return Value - rc

rc (SPLERR) - returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_DestNotFound (2152)	The print device cannot be found.
NERR_DestInvalidState (2162)	This operation cannot be performed on the print device.
NERR_InvalidComputer (2351)	The computer name is invalid.

SplDeleteDevice - Parameters

pszComputerName (PSZ) - input
Name of computer where print device is. to be deleted.

A NULL string specifies the local workstation.

pszPrintDeviceName (PSZ) - input
Name of Print Device.

rc (SPLERR) - returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.

ERROR_NOT_SUPPORTED (50)
This request is not supported by the network.
ERROR_BAD_NETPATH (53)
The network path cannot be located.
NERR_NetNotStarted (2102)
The network program is not started.
NERR_DestNotFound (2152)
The print device cannot be found.
NERR_DestInvalidState (2162)
This operation cannot be performed on the print device.
NERR_InvalidComputer (2351)
The computer name is invalid.

SplDeleteDevice - Remarks

If the print device is currently printing a job, SplDeleteDevice fails and returns NERR_DestInvalidState (2162).

To delete a print device on a remote server requires administrator privilege.

SplDeleteDevice - Related Functions

Related Functions

- [SplCreateDevice](#)
 - [SplEnumDevice](#)
-

SplDeleteDevice - Example Code

This sample code will delete the print device whose name is entered at the prompt.

```
#define INCL_BASE
#define INCL_DOSMEMMGR
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    SPLERR splerr= 0L;
    PSZ    pszComputerName ;
    PSZ    pszPrintDeviceName ;

    /* Check that the parameters were entered at the command line.          */
    if (argc != 2)
    {
        printf("Syntax:  sddel  PrintDeviceName  \n");
        DosExit( EXIT_PROCESS , 0 ) ;
    }

    /* Computer name of NULL indicates the local computer.                  */
    pszComputerName = (PSZ)NULL ;
```

```

/* Set the PrintDeviceName to the value entered at the command line.      */
pszPrintDeviceName = argv[1];

/* Make the call and print out the return code.                          */
splerr=SplDeleteDevice(pszComputerName, pszPrintDeviceName);
switch (splerr)
{
    case NO_ERROR:
        printf("Print Device %s was deleted.\n",pszPrintDeviceName);
        break;
    case NERR_DestNotFound :
        printf("Destination does not exist.\n");
        break;
    case NERR_DestInvalidState:
        printf("This operation can't be performed on the print device.\n");
        break;
    case NERR_SpoolerNotLoaded:
        printf("The Spooler is not running.\n");
        break;
    default:
        printf("SplDeleteDevice Errorcode = %ld\n",splerr);
} /* endswitch */
DosExit( EXIT_PROCESS , 0 ) ;
return (splerr) ;
}

```

SplDeleteDevice - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

SplDeleteJob

SplDeleteJob - Syntax

This function deletes a job from a print queue.

```

#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where job is to be deleted. */
PSZ      pszQueueName;    /* Queue Name. */
ULONG    ulJob;           /* Job identification number. */
SPLERR    rc;             /* Return code. */

rc = SplDeleteJob(pszComputerName, pszQueueName,
    ulJob);

```

SplDeleteJob Parameter - pszComputerName

pszComputerName (PSZ) - input

Name of computer where job is to be deleted.

A NULL string specifies the local workstation.

SplDeleteJob Parameter - pszQueueName

pszQueueName (PSZ) - input

Queue Name.

SplDeleteJob Parameter - ulJob

ulJob (ULONG) - input

Job identification number.

SplDeleteJob Return Value - rc

rc (SPLERR) - returns

Return code.

NO_ERROR (0)

No errors occurred.

ERROR_ACCESS_DENIED (5)

Access is denied.

ERROR_NOT_SUPPORTED (50)

This request is not supported by the network.

ERROR_BAD_NETPATH (53)

The network path cannot be located.

NERR_NetNotStarted (2102)

The network program is not started.

NERR_JobNotFound (2151)

The print job does not exist.

NERR_ProcNoRespond (2160)

The queue processor is not responding.

NERR_SpoolerNotLoaded (2161)

The spooler is not running.

NERR_InvalidComputer (2351)

The computer name is invalid.

SplDeleteJob - Parameters

pszComputerName ([PSZ](#)) - input

Name of computer where job is to be deleted.

A NULL string specifies the local workstation.

pszQueueName ([PSZ](#)) - input

Queue Name.

ulJob ([ULONG](#)) - input

Job identification number.

rc ([SPLERR](#)) - returns

Return code.

NO_ERROR (0)

No errors occurred.

ERROR_ACCESS_DENIED (5)

Access is denied.

ERROR_NOT_SUPPORTED (50)

This request is not supported by the network.

ERROR_BAD_NETPATH (53)

The network path cannot be located.

NERR_NetNotStarted (2102)

The network program is not started.

NERR_JobNotFound (2151)

The print job does not exist.

NERR_ProcNoRespond (2160)

The queue processor is not responding.

NERR_SpoolerNotLoaded (2161)

The spooler is not running.

NERR_InvalidComputer (2351)

The computer name is invalid.

SplDeleteJob - Remarks

It is possible to delete a job that is currently printing.

If the print queue on which the print job is submitted is pending deletion (following a [SplDeleteQueue](#) call), and the print job is the last in the queue, this function has the additional effect of deleting the queue.

A user with administrator privilege can delete any job.

A job created locally can be deleted locally regardless of user privilege level, but can be deleted remotely only by an administrator.

A remote job can be deleted by a user without administrator privilege only if the username of the person initiating the request is the same as the username of the person who created the job.

SplDeleteJob - Related Functions

Related Functions

- [SplCopyJob](#)
- [SplEnumJob](#)
- [SplQueryJob](#)

SplDeleteJob - Example Code

This sample code will delete the job id that is entered at the prompt.

```
#define INCL_BASE
#define INCL_SPL
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>      /* for printf function */
#include <stdlib.h>     /* for atoi function */

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    SPLERR splerr ;
    ULONG ulJob ;
    PSZ    pszComputerName = NULL ;
    PSZ    pszQueueName = NULL ;

    /* Get job id from the input argument. */
    ulJob = atoi(argv[1]);

    /* Call the function to do the delete. If an error is      */
    /* returned, print it.                                     */
    splerr = SplDeleteJob( pszComputerName, pszQueueName, ulJob);

    if (splerr != NO_ERROR)
    {
        switch (splerr)
        {
            case NERR_JobNotFound :
                printf("Job does not exist.\n");
                break;
            case NERR_JobInvalidState:
                printf("This operation can't be performed on the print job.\n");
                break;
            default:
                printf("Errorcode = %ld\n",splerr);
        } /* endswitch */
    }
    else
    {
        printf("Job %d was deleted.\n",ulJob);
    } /* endif */
    DosExit( EXIT_PROCESS , 0 ) ;
    return (splerr);
}
```

SplDeleteJob - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

SplDeletePort

SplDeletePort - Syntax

Description:

This function is called to remove a printer port.

Note: There is currently no support for using this API to remotely remove a printer port.

```
#define INCL_SPL
#define INCL_SPLBIDI
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where port is to be removed. */
PSZ      pszPortName;    /* Name of printer port to remove. */
ULONG    rc;              /* Return codes. */

rc = SplDeletePort(pszComputerName, pszPortName);
```

SplDeletePort Parameter - pszComputerName

pszComputerName (PSZ) - input

Name of computer where port is to be removed.

A NULL string identifies the local workstation. This currently must be NULL.

SplDeletePort Parameter - pszPortName

pszPortName (PSZ) - input

Name of printer port to remove.

This can be a virtual port name. Following is an example of a port name:
LPT5

Note: This port cannot be removed if it is referenced by a print device.

SplDeletePort Return Value - rc

rc (**ULONG**) - returns
Return codes.

0	Success
ERROR_DEVICE_IN_USE(99)	Port name is connected to a print queue.
ERROR_INVALID_PARAMETER(87)	An invalid buffer was given.
ERROR_NOT_SUPPORTED(50)	<i>pszComputerName</i> is not supported.

SplDeletePort - Parameters

pszComputerName (**PSZ**) - input
Name of computer where port is to be removed.

A NULL string identifies the local workstation. This currently must be NULL.

pszPortName (**PSZ**) - input
Name of printer port to remove.

This can be a virtual port name. Following is an example of a port name:
LPT5

Note: This port cannot be removed if it is referenced by a print device.

rc (**ULONG**) - returns
Return codes.

0	Success
ERROR_DEVICE_IN_USE(99)	Port name is connected to a print queue.
ERROR_INVALID_PARAMETER(87)	An invalid buffer was given.
ERROR_NOT_SUPPORTED(50)	<i>pszComputerName</i> is not supported.

SplDeletePort - Remarks

None.

SplDeletePort - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

SplDeleteQueue

SplDeleteQueue - Syntax

This function deletes a print queue from the spooler.

```
#define INCL_SPL /* Or use INCL_PM, */  
#include <os2.h>  
  
PSZ      pszComputerName; /* Name of computer where queue is to be deleted. */  
PSZ      pszQueueName;   /* Queue name. */  
SPLERR   rc;              /* Return code. */  
  
rc = SplDeleteQueue(pszComputerName, pszQueueName);
```

SplDeleteQueue Parameter - pszComputerName

pszComputerName (PSZ) - input
Name of computer where queue is to be deleted.

A NULL string specifies the local workstation.

SplDeleteQueue Parameter - pszQueueName

pszQueueName (PSZ) - input
Queue name.

SplDeleteQueue Return Value - rc

rc (SPLERR) - returns

Return code.

NO_ERROR (0)

No errors occurred.

ERROR_ACCESS_DENIED (5)

Access is denied.

ERROR_NOT_SUPPORTED (50)

This request is not supported by the network.

ERROR_BAD_NETPATH (53)

The network path cannot be located.

ERROR_INVALID_PARAMETER (87)

An invalid parameter is specified.

NERR_NetNotStarted (2102)

The network program is not started.

NERR_QNotFound (2150)

The printer queue does not exist.

NERR_QInvalidState (2163)

This operation cannot be performed on the print queue.

NERR_InvalidComputer (2351)

The computer name is invalid.

SplDeleteQueue - Parameters

pszComputerName (PSZ) - input

Name of computer where queue is to be deleted.

A NULL string specifies the local workstation.

pszQueueName (PSZ) - input

Queue name.

rc (SPLERR) - returns

Return code.

NO_ERROR (0)

No errors occurred.

ERROR_ACCESS_DENIED (5)

Access is denied.

ERROR_NOT_SUPPORTED (50)

This request is not supported by the network.

ERROR_BAD_NETPATH (53)

The network path cannot be located.

ERROR_INVALID_PARAMETER (87)

An invalid parameter is specified.

NERR_NetNotStarted (2102)

The network program is not started.

NERR_QNotFound (2150)

The printer queue does not exist.

NERR_QInvalidState (2163)

This operation cannot be performed on the print queue.

NERR_InvalidComputer (2351)

The computer name is invalid.

SplDeleteQueue - Remarks

If there are print jobs in the queue, SplDeleteQueue marks the queue PRQ3_PENDING. No further jobs can then be added to the queue, which is deleted when all jobs are printed. A queue marked PRQ3_PENDING can be held, and jobs in the queue can be held, restarted, and repeated.

If a queue is held and there are jobs on the queue, a SplDeleteQueue function fails with NERR_QInvalidState (2163).

To delete a queue on a remote server requires administrator privilege on the remote server.

SplDeleteQueue - Related Functions

Related Functions

- [SplCreateQueue](#)
- [SplEnumQueue](#)
- [SplQueryQueue](#)

SplDeleteQueue - Example Code

This sample code will delete the queue name that is entered at the prompt.

```
#define INCL_SPL
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>          /* for printf function */

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    SPLERR splerr ;
    PSZ    pszComputerName = NULL ;
    PSZ    pszQueueName ;

    /* Get queue name from the input argument */
    pszQueueName = argv[1];

    /* Call the function to do the delete. If an error is returned, print it.
*/
    splerr=SplDeleteQueue(pszComputerName, pszQueueName);

    if (splerr != 0L)
    {
        switch (splerr)
        {
            case NERR_QNotFound :
                printf("Queue does not exist.\n");
                break;
            case NERR_QInvalidState:
                printf("This operation can't be performed on the print queue.\n");
                break;
            default:
                printf("Errorcode = %ld\n",splerr);
        } /* endswitch */
    }
    else
    {
        printf("Queue %s was deleted.\n",pszQueueName);
    } /* endif */
    DosExit( EXIT_PROCESS , 0 ) ;
    return (splerr);
}
```

SplDeleteQueue - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

SplDisable

SplDisable - Syntax

Description:

This function is called to disable the OS/2 spooler. The spooler remains disabled across reboots. It can be re-enabled using SplEnable.

Note: There is currently no support for using this API to remotely disable a print spooler.

```
#define INCL_SPL
#define INCL_SPLBIDI
#include <os2.h>

PSZ      pszComputerName; /* Name of the computer on which to disable the spooler. */
PVOID    pReserved;      /* Must be NULL. Reserved for future use. */
ULONG    rc;              /* Return codes. */

rc = SplDisable(pszComputerName, pReserved);
```

SplDisable Parameter - pszComputerName

pszComputerName ([PSZ](#)) - input

Name of the computer on which to disable the spooler.

This value currently must be NULL to disable the local spooler.

SplDisable Parameter - pReserved

pReserved ([PVOID](#)) - input

Must be NULL. Reserved for future use.

SplDisable Return Value - rc

rc ([ULONG](#)) - returns
Return codes.

0	Success
ERROR_INVALID_PARAMETER(87)	An invalid buffer given.
PMERR_SPL_SPOOLER_NOT_INSTALLED(0x4009)	The spooler is not enabled.

SplDisable - Parameters

pszComputerName ([PSZ](#)) - input
Name of the computer on which to disable the spooler.

This value currently must be NULL to disable the local spooler.

pReserved ([PVOID](#)) - input
Must be NULL. Reserved for future use.

rc ([ULONG](#)) - returns
Return codes.

0	Success
ERROR_INVALID_PARAMETER(87)	An invalid buffer given.
PMERR_SPL_SPOOLER_NOT_INSTALLED(0x4009)	The spooler is not enabled.

SplDisable - Remarks

None.

SplDisable - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)

SplDisplayControlPanel

SplDisplayControlPanel - Syntax

Description:

This function displays a control panel for a printer that is communicating bidirectionally.

```
#define INCL_SPL
#define INCL_SPLBIDI
#include <os2.h>

PSZ      pszComputerName; /* Printer server name for displaying a remote printer panel from a client machine. */
PSZ      pszPortName;    /* Printer port name. */
PSZ      pszDeviceID;    /* Device ID, if known, for the printer connected to pszPortName. */
PSZ      pszControlPanel; /* Name of control panel to call to display the front panel for this printer. */
HAB      hab;            /* Anchor block handle. */
ULONG    flCapabilities; /* Capabilities requested. */
ULONG    rc;             /* Return codes. */

rc = SplDisplayControlPanel(pszComputerName,
                           pszPortName, pszDeviceID, pszControlPanel,
                           hab, flCapabilities);
```

SplDisplayControlPanel Parameter - pszComputerName

pszComputerName (PSZ) - input

Printer server name for displaying a remote printer panel from a client machine.

This parameter is NULL for printers whose ports exist on the current machine.

SplDisplayControlPanel Parameter - pszPortName

pszPortName (PSZ) - input

Printer port name.

The name of the printer port for *pszComputerName*, used to display a control panel. Following is an example of a port name:
LPT1

Device ID, if known, for the printer connected to *pszPortName*.

Name of control panel to call to display the front panel for this printer.

- The name registered with `SplRegisterControlPanel`
- One of the control panel names returned by `SplGetControlPanelList`

Anchor block handle.

Capabilities requested.

Set if an administrator wants the control panel.

Return codes.

Success

ERROR_INVALID_NAME(123)

pszName is not the name of a control panel DLL.

ERROR_NOT_SUPPORTED(50)

A control panel for this printer is not supported by the called control panel DLL.

SplDisplayControlPanel - Parameters

pszComputerName ([PSZ](#)) - input

Printer server name for displaying a remote printer panel from a client machine.

This parameter is NULL for printers whose ports exist on the current machine.

pszPortName ([PSZ](#)) - input

Printer port name.

The name of the printer port for *pszComputerName*, used to display a control panel. Following is an example of a port name:
LPT1

pszDeviceID ([PSZ](#)) - input

Device ID, if known, for the printer connected to *pszPortName*.

pszControlPanel ([PSZ](#)) - input

Name of control panel to call to display the front panel for this printer.

This name can be one of the following:

- The name registered with [SplRegisterControlPanel](#)
- One of the control panel names returned by [SplGetControlPanelList](#)

hAb ([HAB](#)) - input

Anchor block handle.

flCapabilities ([ULONG](#)) - input

Capabilities requested.

FL_ADMIN - 0x00000001

Set if an administrator wants the control panel.

rc ([ULONG](#)) - returns

Return codes.

0 Success

ERROR_INVALID_NAME(123)

pszName is not the name of a control panel DLL.

ERROR_NOT_SUPPORTED(50)

A control panel for this printer is not supported by the called control panel DLL.

SplDisplayControlPanel - Remarks

None.

SplDisplayControlPanel - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

SplEnable

SplEnable - Syntax

Description:

This function is called to enable the OS/2 spooler.

Note: There is currently no support for using this API to remotely enable a print spooler.

```
#define INCL_SPL
#define INCL_SPLBIDI
#include <os2.h>

PSZ      pszComputerName; /* Name of the computer on which to enable the spooler. */
PSZ      pszDirectory;    /* Name of the directory containing the spool files. */
PVOID    pReserved;       /* Must be NULL. Reserved for future use. */
ULONG    rc;              /* Return codes. */

rc = SplEnable(pszComputerName, pszDirectory,
               pReserved);
```

SplEnable Parameter - pszComputerName

pszComputerName ([PSZ](#)) - input

Name of the computer on which to enable the spooler.

This value currently must be NULL to enable the local spooler.

SplEnable Parameter - pszDirectory

pszDirectory (PSZ) - input
Name of the directory containing the spool files.

This value must currently be set to NULL to use the spooler directory as defined in the SYSTEM INI file.

SplEnable Parameter - pReserved

pReserved (PVOID) - input
Must be NULL. Reserved for future use.

SplEnable Return Value - rc

rc (ULONG) - returns
Return codes.

0	Success
ERROR_INVALID_PARAMETER(87)	An invalid buffer given.
PMERR_SPL_ALREADY_INITIALIZED(0x4FFD)	The spooler is already enabled.
PMERR_SPL_NO_MEMORY(0x4007)	Not enough memory to enable spooler.

SplEnable - Parameters

pszComputerName (PSZ) - input
Name of the computer on which to enable the spooler.

This value currently must be NULL to enable the local spooler.

pszDirectory (PSZ) - input
Name of the directory containing the spool files.

This value must currently be set to NULL to use the spooler directory as defined in the SYSTEM INI file.

pReserved (PVOID) - input
Must be NULL. Reserved for future use.

rc (ULONG) - returns
Return codes.

0	Success
---	---------

ERROR_INVALID_PARAMETER(87)
An invalid buffer given.

PMERR_SPL_ALREADY_INITIALIZED(0x4FFD)
The spooler is already enabled.

PMERR_SPL_NO_MEMORY(0x4007)
Not enough memory to enable spooler.

SplEnable - Remarks

None.

SplEnable - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

SplEnumDevice

SplEnumDevice - Syntax

This function lists print device on a server, optionally supplying status information.

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where print devices are to. be listed. */
ULONG    ulLevel;         /* Level of detail required. */
PVOID    pBuf;            /* Buffer. */
ULONG    cbBuf;           /* Size, in bytes, of Buffer. */
PULONG   pcReturned;      /* Number of entries returned. */
PULONG   pcTotal;         /* Number of entries available. */
PULONG   pcbNeeded;       /* Size in bytes of available information. */
PVOID    pReserved;       /* Reserved value, must be NULL. */
SPLERR   rc;              /* Return code. */

rc = SplEnumDevice(pszComputerName, ulLevel,
                  pBuf, cbBuf, pcReturned, pcTotal, pcbNeeded,
                  pReserved);
```

SplEnumDevice Parameter - pszComputerName

pszComputerName ([PSZ](#)) - input

Name of computer where print devices are to. be listed.

A NULL string specifies the local workstation.

SplEnumDevice Parameter - ulLevel

ulLevel ([ULONG](#)) - input

Level of detail required.

This must be 0, 2 or 3.

SplEnumDevice Parameter - pBuf

pBuf ([PVOID](#)) - output

Buffer.

The returned contents of the buffer depend on the value indicated in *ulLevel* as follows:

<i>ulLevel</i>	Buffer Contents
0	An array of port names. Each name consists of 9 characters, including a null terminator.
2	An array of print device names of type PSZ .
3	An array of PRDINFO3 structures.

If no job is printing on the print device, bits 2-11 of *fsStatus* in the [PRDINFO3](#) data structure are meaningless.

SplEnumDevice Parameter - cbBuf

cbBuf ([ULONG](#)) - input

Size, in bytes, of Buffer.

It must be greater than 0.

SplEnumDevice Parameter - pcReturned

pcReturned ([PULONG](#)) - output

Number of entries returned.

SplEnumDevice Parameter - pcTotal

pcTotal ([PULONG](#)) - output
Number of entries available.

SplEnumDevice Parameter - pcbNeeded

pcbNeeded ([PULONG](#)) - output
Size in bytes of available information.

A value of 0 specifies that the size is not known.

SplEnumDevice Parameter - pReserved

pReserved ([PVOID](#)) - output
Reserved value, must be NULL.

SplEnumDevice Return Value - rc

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)
No errors occurred.

ERROR_NOT_SUPPORTED (50)
This request is not supported by the network.

ERROR_BAD_NETPATH (53)
The network path cannot be located.

ERROR_INVALID_PARAMETER (87)
An invalid parameter is specified.

ERROR_INVALID_LEVEL (124)
The level parameter is invalid.

ERROR_MORE_DATA (234)
Additional data is available.

NERR_NetNotStarted (2102)
The network program is not started.

NERR_InvalidComputer (2351)
The computer name is invalid.

SplEnumDevice - Parameters

pszComputerName ([PSZ](#)) - input

Name of computer where print devices are to be listed.

A NULL string specifies the local workstation.

ulLevel ([ULONG](#)) - input

Level of detail required.

This must be 0, 2 or 3.

pBuf ([PVOID](#)) - output

Buffer.

The returned contents of the buffer depend on the value indicated in *ulLevel* as follows:

<i>ulLevel</i>	Buffer Contents
0	An array of port names. Each name consists of 9 characters, including a null terminator.
2	An array of print device names of type PSZ .
3	An array of PRDINFO3 structures.

If no job is printing on the print device, bits 2-11 of *fsStatus* in the [PRDINFO3](#) data structure are meaningless.

cbBuf ([ULONG](#)) - input

Size, in bytes, of Buffer.

It must be greater than 0.

pcReturned ([PULONG](#)) - output

Number of entries returned.

pcTotal ([PULONG](#)) - output

Number of entries available.

pcbNeeded ([PULONG](#)) - output

Size in bytes of available information.

A value of 0 specifies that the size is not known.

pReserved ([PVOID](#)) - output

Reserved value, must be NULL.

rc ([SPLERR](#)) - returns

Return code.

NO_ERROR (0)

No errors occurred.

ERROR_NOT_SUPPORTED (50)

This request is not supported by the network.

ERROR_BAD_NETPATH (53)

The network path cannot be located.

ERROR_INVALID_PARAMETER (87)

An invalid parameter is specified.

ERROR_INVALID_LEVEL (124)

The level parameter is invalid.

ERROR_MORE_DATA (234)

Additional data is available.

NERR_NetNotStarted (2102)

The network program is not started.

NERR_InvalidComputer (2351)

The computer name is invalid.

SplEnumDevice - Remarks

If *ulLevel* is set to 0, each port name in *pBuf* is truncated to 8 characters.

SplEnumDevice - Related Functions

Related Functions

- [SplCreateDevice](#)
 - [SplDeleteDevice](#)
-

SplEnumDevice - Example Code

This sample code enumerates all the devices on the local workstation. It then prints out the information.

```
#define INCL_BASE
#define INCL_DOSMEMMGR
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>

INT main ()
{
    ULONG    cbBuf ;
    ULONG    cTotal;
    ULONG    cReturned ;
    ULONG    cbNeeded ;
    ULONG    ulLevel = 3L;
    ULONG    i ;
    SPLERR    splerr ;
    PSZ      pszComputerName ;
    PBYTE    pBuf ;
    PPRDINFO3 pprd3 ;

    pszComputerName = (PSZ)NULL ;

    /* Make the call with cBuf = 0 so that you will get the size of the */
    /* buffer needed returned in cbNeeded.                               */
    splerr = SplEnumDevice(pszComputerName, ulLevel, pBuf, 0L, /* cbBuf */
                          &cReturned, &cTotal, &cbNeeded,
                          NULL) ;

    /* Only continue if the error codes ERROR_MORE_DATA or             */
    /* NERR_BufTooSmall are returned.                                   */
    if (splerr == ERROR_MORE_DATA || splerr == NERR_BufTooSmall)
    {
        /* Allocate memory for the buffer that will hold the returning info. */
        if (!DosAllocMem( &pBuf, cbNeeded,
                        PAG_READ|PAG_WRITE|PAG_COMMIT) )
        {
            cbBuf = cbNeeded ;

            /* Make call again with the proper buffer size.              */
            splerr = SplEnumDevice(pszComputerName, ulLevel, pBuf, cbBuf,
                                  &cReturned, &cTotal,
                                  &cbNeeded, NULL) ;

            /* If no errors, print out the buffer information.            */
            if (splerr == NO_ERROR)
            {
                for (i=0; i < cReturned ; i++)
                {
```

```

        /* Each time through the loop increase the pointer. */
        pprd3 = (PPRDINFO3)pBuf+i ;
        printf("Device info:pszPrinterName - %s\n",
               pprd3->pszPrinterName) ;
        printf("  pszUserName - %s\n", pprd3->pszUserName);
        printf("  pszLogAddr - %s\n", pprd3->pszLogAddr);
        printf("  uJobId      - %d  fsStatus - %X\n",
               pprd3->uJobId , pprd3->fsStatus);
        printf("  pszStatus   - %s\n", pprd3->pszStatus);
        printf("  pszComment  - %s\n", pprd3->pszComment);
        printf("  pszDrivers  - %s\n", pprd3->pszDrivers);
        printf("  time        - %d usTimeOut - %X\n",
               pprd3->time , pprd3->usTimeOut);
    }
    }
    DosFreeMem(pBuf) ;
}
} /* end if */
else
{
    printf("SplEnumDevice splerr=%ld, cTotal=%ld, cReturned=%ld, cbNeeded=%ld\n",
           splerr, cTotal, cReturned, cbNeeded) ;
}
DosExit( EXIT_PROCESS , 0 ) ;
return(splerr);
} /* end main */

```

SplEnumDevice - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

SplEnumDriver

SplEnumDriver - Syntax

This function lists printer presentation drivers on the local workstation or on a remote server.

```

#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where queues are to be listed. */
ULONG    ulLevel;         /* Level of detail. */
PVOID    pBuf;           /* Buffer. */
ULONG    cbBuf;           /* Size, in bytes, of Buffer. */
PULONG   pcReturned;      /* Number of entries returned. */
PULONG   pcTotal;         /* Total number of entries available. */
PULONG   pcbNeeded;       /* Size in bytes of available information. */

```



```

PVOID      pReserved;          /* Reserved value, must be NULL. */
SPLERR     rc;                 /* Return code. */

rc = SplEnumDriver(pszComputerName, ulLevel,
                  pBuf, cbBuf, pcReturned, pcTotal, pcbNeeded,
                  pReserved);

```

SplEnumDriver Parameter - pszComputerName

pszComputerName ([PSZ](#)) - input
 Name of computer where queues are to be listed.

A NULL string specifies the local workstation.

SplEnumDriver Parameter - ulLevel

ulLevel ([ULONG](#)) - input
 Level of detail.

The level of detail required. This must be 0.

SplEnumDriver Parameter - pBuf

pBuf ([PVOID](#)) - output
 Buffer.

The returned contents in the buffer are:

	<i>ulLevel</i>	Buffer Contents
0		An array of PRDRIVINFO structures

SplEnumDriver Parameter - cbBuf

cbBuf ([ULONG](#)) - input
 Size, in bytes, of Buffer.

It must be greater than 0.

SplEnumDriver Parameter - pcReturned

pcReturned ([PULONG](#)) - output
Number of entries returned.

SplEnumDriver Parameter - pcTotal

pcTotal ([PULONG](#)) - output
Total number of entries available.

SplEnumDriver Parameter - pcbNeeded

pcbNeeded ([PULONG](#)) - output
Size in bytes of available information.

A value of 0 specifies that the size is not known.

SplEnumDriver Parameter - pReserved

pReserved ([PVOID](#)) - output
Reserved value, must be NULL.

SplEnumDriver Return Value - rc

rc ([SPLERR](#)) - returns
Return code.

- NO_ERROR** (0)
No errors occurred.
- ERROR_ACCESS_DENIED** (5)
Access is denied.
- ERROR_NOT_SUPPORTED** (50)
This request is not supported by the network.
- ERROR_BAD_NETPATH** (53)
The network path cannot be located.
- ERROR_INVALID_PARAMETER** (87)
An invalid parameter is specified.
- ERROR_INVALID_LEVEL** (124)
The level parameter is invalid.
- ERROR_MORE_DATA** (234)
Additional data is available.
- NERR_NetNotStarted** (2102)
The network program is not started.

NERR_BufTooSmall (2123)
The API return buffer is too small.
NERR_InvalidComputer (2351)
The computer name is invalid.

SplEnumDriver - Parameters

pszComputerName ([PSZ](#)) - input
Name of computer where queues are to be listed.

A NULL string specifies the local workstation.

ulLevel ([ULONG](#)) - input
Level of detail.

The level of detail required. This must be 0.

pBuf ([PVOID](#)) - output
Buffer.

The returned contents in the buffer are:

	<i>ulLevel</i>	Buffer Contents
0		An array of PRDRIVINFO structures

cbBuf ([ULONG](#)) - input
Size, in bytes, of Buffer.

It must be greater than 0.

pcReturned ([PULONG](#)) - output
Number of entries returned.

pcTotal ([PULONG](#)) - output
Total number of entries available.

pcbNeeded ([PULONG](#)) - output
Size in bytes of available information.

A value of 0 specifies that the size is not known.

pReserved ([PVOID](#)) - output
Reserved value, must be NULL.

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
ERROR_MORE_DATA (234)	Additional data is available.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_BufTooSmall (2123)	The API return buffer is too small.

NERR_InvalidComputer (2351)
The computer name is invalid.

SplEnumDriver - Related Functions

Related Functions

- [SplCreateDevice](#)
 - [SplCreateQueue](#)
 - [SplSetDevice](#)
 - [SplSetQueue](#)
-

SplEnumDriver - Example Code

This sample code will enumerate all the drivers on a local computer.

```
#define INCL_BASE
#define INCL_DOSMEMMGR
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>          /* for printf function */

INT main ()
{
    SPLERR splerr ;
    ULONG  cbBuf ;
    ULONG  cTotal ;
    ULONG  cReturned ;
    ULONG  cbNeeded ;
    ULONG  i ;
    PSZ    pszComputerName = NULL ;
    PSZ    pszDriverName ;
    PBYTE  pbuf ;

    /* Call the function the first time with zero in cbBuf. The count of bytes */
    /* needed for the buffer to hold all the info will be returned in cbNeeded.*/
    splerr = SplEnumDriver(pszComputerName, 0L, NULL, 0L,
                          &cReturned, &cTotal, &cbNeeded,
                          NULL );

    /* If the return code is ERROR_MORE_DATA or NERR_BufTooSmall, then all the */
    /* parameters were correct; and we can continue. */
    if (splerr == ERROR_MORE_DATA || splerr == NERR_BufTooSmall)
    {
        /* Allocate memory for the buffer to hold the returned information. Use */
        /* the count of bytes that were returned by our first call. */
        if (!DosAllocMem( &pbuf, cbNeeded,
                        PAG_READ|PAG_WRITE|PAG_COMMIT) )
        {
            /* Set count of bytes to the value returned by our first call. */
            cbBuf= cbNeeded ;

            /* Now call the function a second time with the correct values, and */
            /* the information will be returned in the buffer. */
            splerr= SplEnumDriver(pszComputerName, 0L, pbuf, cbBuf,
                                &cReturned ,&cTotal, &cbNeeded,
                                NULL ) ;

            if (splerr == NO_ERROR)
            {
                /* Set a pointer to point to the beginning of the buffer. */
                pszDriverName = (PSZ)pbuf;
            }
        }
    }
}
```

```

        /* Print the names that are in the buffer. The count of the number*/
        /* of names in pBuf have been returned in cReturned.          */
        for (i=0;i < cReturned ; i++)
        {
            printf("Driver name - %s\n", pszDriverName) ;
            /* Increment the pointer to point to the next name.        */
            pszDriverName += DRV_NAME_SIZE + DRV_DEVICENAME_SIZE + 2;
        }
        /* Free the memory allocated for the buffer.                  */
        DosFreeMem(pbuf) ;
    }
}
else
{
    /* If the first call to the function returned any error code other */
    /* than ERROR_MORE_DATA or NERR_BufTooSmall, we print the following. */
    printf("SplEnumDriver error=%ld \n",splerr) ;
}
DosExit( EXIT_PROCESS , 0 ) ;
return (splerr);
}

```

SplEnumDriver - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

SplEnumJob

SplEnumJob - Syntax

This function lists the jobs in a print queue, optionally supplying status information on each job.

```

#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where jobs are to be listed. */
PSZ      pszQueueName;   /* Queue name. */
ULONG    ulLevel;        /* Level of detail required. */
PVOID    pBuf;           /* Buffer. */
ULONG    cbBuf;          /* Size, in bytes, of Buffer. */
PULONG   pcReturned;     /* Number of entries returned. */
PULONG   pcTotal;        /* Number of entries available. */
PULONG   pcbNeeded;      /* Size in bytes of available information. */
PVOID    pReserved;      /* Reserved vaue, must be NULL. */
SPLERR   rc;             /* Return code. */

```

```
rc = SplEnumJob(pszComputerName, pszQueueName,
               ulLevel, pBuf, cbBuf, pcReturned, pcTotal,
               pcbNeeded, pReserved);
```

SplEnumJob Parameter - pszComputerName

pszComputerName ([PSZ](#)) - input
Name of computer where jobs are to be listed.

A NULL string specifies the local workstation.

SplEnumJob Parameter - pszQueueName

pszQueueName ([PSZ](#)) - input
Queue name.

SplEnumJob Parameter - ulLevel

ulLevel ([ULONG](#)) - input
Level of detail required.

This must be 0 or 2.

SplEnumJob Parameter - pBuf

pBuf ([PVOID](#)) - output
Buffer.

The returned contents in the buffer depend on the value specified in *ulLevel* as follows:

<i>ulLevel</i>	Buffer Contents
0	An array containing a <i>uJobId</i> for each of <i>pcReturned</i> jobs.
2	An array containing a PRJINFO2 structure for each of <i>pcReturned</i> jobs.

SplEnumJob Parameter - cbBuf

cbBuf ([ULONG](#)) - input
Size, in bytes, of Buffer.

It must be greater than 0.

SplEnumJob Parameter - pcReturned

pcReturned ([PULONG](#)) - output
Number of entries returned.

SplEnumJob Parameter - pcTotal

pcTotal ([PULONG](#)) - output
Number of entries available.

SplEnumJob Parameter - pcbNeeded

pcbNeeded ([PULONG](#)) - output
Size in bytes of available information.

A value of 0 specifies that the size is not known.

SplEnumJob Parameter - pReserved

pReserved ([PVOID](#)) - output
Reserved vaue, must be NULL.

SplEnumJob Return Value - rc

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)
No errors occurred.
ERROR_NOT_SUPPORTED (50)
This request is not supported by the network.

ERROR_INVALID_PARAMETER (87)
 An invalid parameter is specified.
 ERROR_INVALID_LEVEL (124)
 The level parameter is invalid.
 ERROR_MORE_DATA (234)
 Additional data is available.
 NERR_NetNotStarted (2102)
 The network program is not started.
 NERR_QNotFound (2150)
 The printer queue does not exist.
 NERR_SpoolerNotLoaded (2161)
 The spooler is not running.
 NERR_InvalidComputer (2351)
 The computer name is invalid.

SplEnumJob - Parameters

pszComputerName ([PSZ](#)) - input
 Name of computer where jobs are to be listed.

 A NULL string specifies the local workstation.

pszQueueName ([PSZ](#)) - input
 Queue name.

ulLevel ([ULONG](#)) - input
 Level of detail required.

 This must be 0 or 2.

pBuf ([PVOID](#)) - output
 Buffer.

The returned contents in the buffer depend on the value specified in *ulLevel* as follows:

<i>ulLevel</i>	Buffer Contents
0	An array containing a <i>uJobId</i> for each of <i>pcReturned</i> jobs.
2	An array containing a PRJINFO2 structure for each of <i>pcReturned</i> jobs.

cbBuf ([ULONG](#)) - input
 Size, in bytes, of Buffer.

 It must be greater than 0.

pcReturned ([PULONG](#)) - output
 Number of entries returned.

pcTotal ([PULONG](#)) - output
 Number of entries available.

pcbNeeded ([PULONG](#)) - output
 Size in bytes of available information.

 A value of 0 specifies that the size is not known.

pReserved ([PVOID](#)) - output
 Reserved vaue, must be NULL.

rc ([SPLERR](#)) - returns
 Return code.

NO_ERROR (0)
 No errors occurred.
 ERROR_NOT_SUPPORTED (50)
 This request is not supported by the network.

ERROR_INVALID_PARAMETER (87)
 An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)
 The level parameter is invalid.
ERROR_MORE_DATA (234)
 Additional data is available.
NERR_NetNotStarted (2102)
 The network program is not started.
NERR_QNotFound (2150)
 The printer queue does not exist.
NERR_SpoolerNotLoaded (2161)
 The spooler is not running.
NERR_InvalidComputer (2351)
 The computer name is invalid.

SplEnumJob - Related Functions

Related Functions

- [SplCopyJob](#)
 - [SplDeleteJob](#)
 - [SplQueryJob](#)
-

SplEnumJob - Example Code

This sample code accepts a queue name from the command line, and then prints out all the information associated with each job in that queue. Level 0 and 2 are valid; we have chosen to print out level 2 information.

```
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>      /* for printf function */

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    ULONG    splerr ;
    ULONG    cbBuf  ;
    ULONG    cTotal ;
    ULONG    cReturned ;
    ULONG    cbNeeded ;
    ULONG    ulLevel;
    ULONG    i ;
    PSZ      pszComputerName ;
    PSZ      pszQueueName ;
    PVOID    pBuf = NULL;
    PPRJINFO2 pprj2 ;

    /* Check that the command line entry was two parameters.          */
    if (argc != 2)
    {
        printf("Syntax:  enumjob QueueName\n");
        DosExit( EXIT_PROCESS , 0 ) ;
    }
    /* Either a NULL or a pointer to a NULL specify the local workstation. */
    pszComputerName = (PSZ)NULL ;

    /* Set queue name equal to the value entered at the command line.    */
    pszQueueName = argv[1];
```

```

/* Valid level are 0 and 2. Level 2 gives info for a PRJINFO2 structure. */
ulLevel = 2L;

/* Make the call the first time with cbBuf = zero so that we can get a */
/* return of the number of bytes that are need for pBuf to hold all of */
/* the information. The bytes needed will be returned in cbNeeded. */
splerr = SplEnumJob(pszComputerName,pszQueueName, ulLevel, pBuf,0L,
                  &cReturned, &cTotal,
                  &cbNeeded, NULL) ;

/* Check that the return code is one of the two valid errors at this time. */
if (splerr == ERROR_MORE_DATA || splerr == NERR_BufTooSmall )
{
    /* Allocate memory for pBuf. ( No error checking is done on DosAllocMem */
    /* call to keep this sample code simple.) */
    DosAllocMem( &pBuf, cbNeeded,
                PAG_READ|PAG_WRITE|PAG_COMMIT );

    /* Set bytes needed for buffer to the value returned by the first call. */
    cbBuf = cbNeeded ;

    /* Make the call with all the valid information. */
    SplEnumJob(pszComputerName,pszQueueName, ulLevel,
                pBuf, cbBuf, &cReturned,&cTotal,
                &cbNeeded,NULL );

    /* Set up a pointer to point to the beginning of the buffer in which we */
    /* have the returned information */
    pprj2=(PPRJINFO2)pBuf;

    /* The number of structures in the buffer(pBuf) are returned in cReturned*/
    /* Implement a for loop to print out the information for each structure. */
    for (i=0; i<cReturned ;i++ )
    {
        printf("Job ID      = %d\n",  pprj2->uJobId);
        printf("Job Priority = %d\n",  pprj2->uPriority);
        printf("User Name   = %s\n",  pprj2->pszUserName);
        printf("Position   = %d\n",  pprj2->uPosition);
        printf("Status      = %d\n",  pprj2->fsStatus);
        printf("Submitted  = %ld\n", pprj2->ulSubmitted);
        printf("Size        = %ld\n", pprj2->ulSize);
        printf("Comment    = %s\n",  pprj2->pszComment);
        printf("Document   = %s\n\n",pprj2->pszDocument);

        /* Increment the pointer to point to the next structure in the buffer*/
        pprj2++;
    } /* endfor */
    /* Free the memory that we allocated to make the call. */
    DosFreeMem(pBuf) ;
}
else
{
    /* If any other error other than ERROR_MORE_DATA or NERR_BufTooSmall, then */
    /* print the returned information. */
    printf("SplEnumJob Error=%ld, Total Jobs=%ld, Returned Jobs=%ld, Bytes Needed=%ld\n",
          splerr, cTotal, cReturned, cbNeeded) ;
}
DosExit( EXIT_PROCESS , 0 ) ;
return (splerr);
}

```

SplEnumJob - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

SplEnumPort

SplEnumPort - Syntax

This function lists printer ports on the local workstation or on a remote server.

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where queues are to be listed. */
ULONG    ulLevel;         /* Level of detail. */
PVOID    pBuf;            /* Buffer. */
ULONG    cbBuf;           /* Size, in bytes, of Buffer. */
PULONG    pcReturned;     /* Number of entries returned. */
PULONG    pcTotal;        /* Total number of entries available. */
PULONG    pcbNeeded;      /* Size in bytes of available information. */
PVOID    pReserved;      /* Reserved. */
SPLERR    rc;             /* Return code. */

rc = SplEnumPort(pszComputerName, ulLevel,
                pBuf, cbBuf, pcReturned, pcTotal, pcbNeeded,
                pReserved);
```

SplEnumPort Parameter - pszComputerName

pszComputerName (**PSZ**) - input
Name of computer where queues are to be listed.

A NULL string specifies the local workstation.

SplEnumPort Parameter - ulLevel

ulLevel (**ULONG**) - input
Level of detail.

The level of detail required. This must be 0 or 1.

SplEnumPort Parameter - pBuf

pBuf (**PVOID**) - output
Buffer.

The returned content in the buffer depends on the value specified in *uLevel* as follows:

	<i>uLevel</i>	Buffer Contents
0		An array of PRPORTINFO structures
1		An array of PRPORTINFO1 structures

SplEnumPort Parameter - cbBuf

cbBuf (**ULONG**) - input
Size, in bytes, of Buffer.

SplEnumPort Parameter - pcReturned

pcReturned (**PULONG**) - output
Number of entries returned.

SplEnumPort Parameter - pcTotal

pcTotal (**PULONG**) - output
Total number of entries available.

SplEnumPort Parameter - pcbNeeded

pcbNeeded (**PULONG**) - output
Size in bytes of available information.

A value of 0 specifies that the size is not known.

SplEnumPort Parameter - pReserved

pReserved ([PVOID](#)) - output
Reserved.

This must be NULL.

SplEnumPort Return Value - rc

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)
No errors occurred.

ERROR_ACCESS_DENIED (5)
Access is denied.

ERROR_NOT_SUPPORTED (50)
This request is not supported by the network.

ERROR_BAD_NETPATH (53)
The network path cannot be located.

ERROR_INVALID_PARAMETER (87)
An invalid parameter is specified.

ERROR_INVALID_LEVEL (124)
The level parameter is invalid.

ERROR_MORE_DATA (234)
Additional data is available.

NERR_NetNotStarted (2102)
The network program is not started.

NERR_BufTooSmall (2123)
The API return buffer is too small.

NERR_InvalidComputer (2351)
The computer name is invalid.

SplEnumPort - Parameters

pszComputerName ([PSZ](#)) - input
Name of computer where queues are to be listed.

A NULL string specifies the local workstation.

ulLevel ([ULONG](#)) - input
Level of detail.

The level of detail required. This must be 0 or 1.

pBuf ([PVOID](#)) - output
Buffer.

The returned content in the buffer depends on the value specified in *ulLevel* as follows:

<i>ulLevel</i>	Buffer Contents
0	An array of PRPORTINFO structures
1	An array of PRPORTINFO1 structures

cbBuf ([ULONG](#)) - input
Size, in bytes, of Buffer.

pcReturned ([PULONG](#)) - output
Number of entries returned.

pcTotal ([PULONG](#)) - output
Total number of entries available.

pcbNeeded ([PULONG](#)) - output
Size in bytes of available information.

A value of 0 specifies that the size is not known.

pReserved ([PVOID](#)) - output
Reserved.

This must be NULL.

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)
No errors occurred.
ERROR_ACCESS_DENIED (5)
Access is denied.
ERROR_NOT_SUPPORTED (50)
This request is not supported by the network.
ERROR_BAD_NETPATH (53)
The network path cannot be located.
ERROR_INVALID_PARAMETER (87)
An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)
The level parameter is invalid.
ERROR_MORE_DATA (234)
Additional data is available.
NERR_NetNotStarted (2102)
The network program is not started.
NERR_BufTooSmall (2123)
The API return buffer is too small.
NERR_InvalidComputer (2351)
The computer name is invalid.

SplEnumPort - Related Functions

Related Functions

- [SplCreateDevice](#)
- [SplSetDevice](#)

SplEnumPort - Example Code

This sample code will print out all the ports and associated information. This is done at level 1, and for the local workstation.

```
#define INCL_DOSMEMMGR
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>

INT main ()
{
    SPLERR splerr ;
    ULONG   cbBuf ;
    ULONG   cTotal;
```

```

ULONG   cReturned ;
ULONG   cbNeeded ;
ULONG   ulLevel = 1;
ULONG   i ;
PSZ     pszComputerName = NULL;
PVOID   pbuf ;
PPRPORTINFO1 pPort1 ;

splerr = SplEnumPort(pszComputerName, ulLevel, pbuf, 0L, /* cbBuf */
                    &cReturned, &cTotal,
                    &cbNeeded, NULL) ;

if (splerr == ERROR_MORE_DATA || NERR_BufTooSmall )
{
    if (!DosAllocMem( &pbuf, cbNeeded,
                     PAG_READ|PAG_WRITE|PAG_COMMIT) )
    {
        cbBuf = cbNeeded ;
        splerr = SplEnumPort(pszComputerName, ulLevel, pbuf, cbBuf,
                            &cReturned, &cTotal,
                            &cbNeeded, NULL) ;

        if (splerr == 0L)
        {
            pPort1 = (PPRPORTINFO1)pbuf ;
            printf("Port names: ");
            for (i=0; i < cReturned; i++)
            {
                printf("Port - %s, Driver - %s Path - %s\n",
                       pPort1->pszPortName, pPort1->pszPortDriverName,
                       pPort1->pszPortDriverPathName ) ;
                pPort1++ ;
            }
            printf("\n");
            DosFreeMem(pbuf) ;
        }
    }
    else
    {
        printf("SplEnumPort splerr=%ld, \n",splerr) ;
    }
    DosExit( EXIT_PROCESS , 0 ) ;
    return (splerr);
} /* end main */

```

SplEnumPort - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

SplEnumPrinter

SplEnumPrinter - Syntax

This function lists print destinations in the system.

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      PSZComputerName; /* Name of computer where queues are to be listed. */
ULONG    ulLevel;         /* Level of detail required. */
ULONG    flType;          /* Type of print destinations required. */
PVOID    pBuf;            /* Buffer. */
ULONG    cbBuf;           /* Size, in bytes, of Buffer. */
PULONG    pcReturned;     /* Number of entries returned. */
PULONG    pcTotal;        /* Number of entries available. */
PULONG    pcbNeeded;      /* Size in bytes of available information. */
PVOID    pReserved;       /* Reserved value, must be NULL. */
SPLERR    rc;             /* Return code. */

rc = SplEnumPrinter(PSZComputerName, ulLevel,
                    flType, pBuf, cbBuf, pcReturned, pcTotal,
                    pcbNeeded, pReserved);
```

SplEnumPrinter Parameter - PSZComputerName

PSZComputerName ([PSZ](#)) - input
Name of computer where queues are to be listed.

This must be NULL.

SplEnumPrinter Parameter - ulLevel

ulLevel ([ULONG](#)) - input
Level of detail required.

This must be 0.

SplEnumPrinter Parameter - flType

flType ([ULONG](#)) - input
Type of print destinations required.

SPL_PR_QUEUE
Return only queues
SPL_PR_DIRECT_DEVICE
Return only direct print devices
SPL_PR_QUEUED_DEVICE
Return only queued print devices
SPL_PR_LOCAL_ONLY
Return only local print destinations

SplEnumPrinter Parameter - pBuf

pBuf ([PVOID](#)) - output
Buffer.

The returned contents in the buffer are as follows:

<i>ulLevel</i>	Buffer Contents
0	An array of PRINTERINFO structures.

When the names of print destinations are returned, calls can be made to [SplQueryQueue](#) or [SplQueryDevice](#) to find out further information about the print destination.

SplEnumPrinter Parameter - cbBuf

cbBuf ([ULONG](#)) - input
Size, in bytes, of Buffer.

It must be greater than 0.

SplEnumPrinter Parameter - pcReturned

pcReturned ([PULONG](#)) - output
Number of entries returned.

SplEnumPrinter Parameter - pcTotal

pcTotal ([PULONG](#)) - output
Number of entries available.

SplEnumPrinter Parameter - pcbNeeded

pcbNeeded ([PULONG](#)) - output
Size in bytes of available information.

A value of 0 specifies that the size is not known.

SplEnumPrinter Parameter - pReserved

pReserved ([PVOID](#)) - output
Reserved value, must be NULL.

SplEnumPrinter Return Value - rc

rc ([SPLERR](#)) - returns
Return code.

- NO_ERROR** (0)
No errors occurred.
- ERROR_NOT_SUPPORTED** (50)
This request is not supported by the network.
- ERROR_INVALID_PARAMETER** (87)
An invalid parameter is specified.
- ERROR_INVALID_LEVEL** (124)
The level parameter is invalid.
- ERROR_MORE_DATA** (234)
Additional data is available.
- NERR_NetNotStarted** (2102)
The network program is not started.
- NERR_BufTooSmall** (2123)
The API return buffer is too small.

SplEnumPrinter - Parameters

PSZComputerName ([PSZ](#)) - input
Name of computer where queues are to be listed.

This must be NULL.

ulLevel ([ULONG](#)) - input
Level of detail required.

This must be 0.

flType ([ULONG](#)) - input
Type of print destinations required.

- SPL_PR_QUEUE**
Return only queues
- SPL_PR_DIRECT_DEVICE**
Return only direct print devices
- SPL_PR_QUEUED_DEVICE**
Return only queued print devices
- SPL_PR_LOCAL_ONLY**
Return only local print destinations

pBuf ([PVOID](#)) - output
Buffer.

The returned contents in the buffer are as follows:

<i>ulLevel</i>	Buffer Contents
0	An array of PRINTERINFO structures.

When the names of print destinations are returned, calls can be made to [SplQueryQueue](#) or [SplQueryDevice](#) to find out further information about the print destination.

cbBuf ([ULONG](#)) - input
Size, in bytes, of Buffer.

It must be greater than 0.

pcReturned ([PULONG](#)) - output
Number of entries returned.

pcTotal ([PULONG](#)) - output
Number of entries available.

pcbNeeded ([PULONG](#)) - output
Size in bytes of available information.

A value of 0 specifies that the size is not known.

pReserved ([PVOID](#)) - output
Reserved value, must be NULL.

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
ERROR_MORE_DATA (234)	Additional data is available.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_BufTooSmall (2123)	The API return buffer is too small.

SplEnumPrinter - Related Functions

Related Functions

- [SplQueryDevice](#)
 - [SplQueryQueue](#)
-

SplEnumPrinter - Example Code

This example code will print out all queues and printers for the local computer. It will print out both printers that are attached to a queue, and those that are direct printers.

```
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS
```

```

#include <os2.h>
#include <stdio.h>      /* for printf function */

INT main ()
{
    PVOID pBuf;
    ULONG fsType ;
    ULONG cbBuf ;
    ULONG cRes ;
    ULONG cTotal ;
    ULONG cbNeeded ;
    SPLERR splerr = 0 ;
    PPRINTERINFO pRes ;

    /* Set fsType to use all the flags. We will print out local device/queues.*/
    fsType = SPL_PR_QUEUE | SPL_PR_DIRECT_DEVICE |
              SPL_PR_QUEUED_DEVICE | SPL_PR_LOCAL_ONLY;

    /* Make function call with cbBuf equal to zero to get a return in cbNeeded*/
    /* of the number of bytes needed for buffer to hold all the information */

    splerr = SplEnumPrinter ( NULL,0 ,fsType ,NULL ,NULL ,&cRes ,
                              &cTotal,&cbNeeded ,NULL ) ;

    /* The error return code will be one of the two following codes if      */
    /* all the parameters were correct.  Otherwise it could be              */
    /* ERROR_INVALID_PARAMETER.                                             */

    if ( splerr == ERROR_MORE_DATA || splerr == NERR_BufTooSmall )
    {
        /* Allocate memory for the buffer using the count of bytes that were */
        /* returned in cbNeeded.  For simplicity, no error checking is done.  */
        DosAllocMem( &pBuf, cbNeeded,
                     PAG_READ|PAG_WRITE|PAG_COMMIT);

        /* Set count of bytes in buffer to value used to allocate buffer.    */
        cbBuf = cbNeeded;

        /* Call function again with the correct buffer size.                  */
        splerr = SplEnumPrinter ( NULL,0 ,fsType ,pBuf ,cbBuf ,&cRes ,
                                 &cTotal,&cbNeeded,NULL);

        /* If there are any returned structures in the buffer, then we will  */
        /* print out some of the information.                                  */
        if (cRes)
        {
            pRes = (PPRINTERINFO)pBuf ;
            while ( cRes-- )
            {
                /* Look at the flType element in the pRes structure to determine */
                /* what type of print destination the structure represents.        */
                switch (pRes[cRes].flType)
                {
                    case SPL_PR_QUEUE:
                        printf("Print destination %s is a queue.\n",
                               pRes[cRes].pszPrintDestinationName) ;
                        break;
                    case SPL_PR_QUEUED_DEVICE:
                        printf("Print destination %s is a queued printer.\n",
                               pRes[cRes].pszPrintDestinationName) ;
                        break;
                    case SPL_PR_DIRECT_DEVICE:
                        printf("Print destination %s is a direct printer.\n",
                               pRes[cRes].pszPrintDestinationName) ;
                }
                printf("Description -\n\t%s\n\n",pRes[cRes].pszDescription) ;
            }
        }
        DosFreeMem(pBuf);
    }
    else
    {
        /* If we had any other return code other than ERROR_MORE_DATA or */
        /* NERR_BufTooSmall, we will print out the following information.  */
        printf("SplEnumPrinter error= %ld \n",splerr);
    }
    DosExit( EXIT_PROCESS , 0 ) ;
    return (splerr);
}

```

```
}
```

SplEnumPrinter - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

SplEnumQueue

SplEnumQueue - Syntax

This function lists print queues on the local workstation or on a remote server, optionally supplying additional information.

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where queues are to be listed. */
ULONG    ulLevel;         /* Level of detail. */
PVOID    pBuf;            /* Buffer. */
ULONG    cbBuf;           /* Size, in bytes, of Buffer. */
PULONG    pcReturned;     /* Number of entries returned. */
PULONG    pcTotal;        /* Total number of entries available. */
PULONG    pcbNeeded;      /* Size in bytes of available information. */
PVOID     pReserved;      /* Reserved value, must be NULL. */
SPLERR    rc;             /* Return code. */

rc = SplEnumQueue(pszComputerName, ulLevel,
                  pBuf, cbBuf, pcReturned, pcTotal, pcbNeeded,
                  pReserved);
```

SplEnumQueue Parameter - pszComputerName

pszComputerName ([PSZ](#)) - input

Name of computer where queues are to be listed.

A NULL string specifies the local workstation.

SplEnumQueue Parameter - ulLevel

ulLevel ([ULONG](#)) - input
Level of detail.

The level of detail required. This must be 3, 4, 5, 6 or 7.

SplEnumQueue Parameter - pBuf

pBuf ([PVOID](#)) - output
Buffer.

The returned contents in the buffer depend on the value specified in *ulLevel* as follows:

<i>ulLevel</i>	Buffer Contents
3	An array of PRQINFO3 structures. The <i>IsType</i> bit PRQ3_TYPE_APPDEFAULT is set for the application default queue only.
4	An array of <i>*pcReturned</i> PRQINFO3 structures in which each PRQINFO3 structure is followed by an array of PRJINFO2 structures, one for each job in the queue. <i>cJobs</i> in the PRQINFO3 structure gives the number of jobs in the array.
5	An array of PSZ, each pointing to a queue name.
6	An array of PRQINFO6 structures
7	An array of <i>*pcReturned</i> PRQINFO3 structures in which each PRQINFO3 structure is followed by an array of PRJINFO4 structures, one for each job in the queue. <i>cJobs</i> in the PRQINFO3 structure gives the number of jobs in the array.

SplEnumQueue Parameter - cbBuf

cbBuf ([ULONG](#)) - input
Size, in bytes, of Buffer.

It must be greater than 0. Some systems do not allow a size greater than 65535.

SplEnumQueue Parameter - pcReturned

pcReturned ([PULONG](#)) - output
Number of entries returned.

SplEnumQueue Parameter - pcTotal

pcTotal ([PULONG](#)) - output
Total number of entries available.

SplEnumQueue Parameter - pcbNeeded

pcbNeeded ([PULONG](#)) - output
Size in bytes of available information.

A value of 0 specifies that the size is not known. This can occur when enumerating queues on a print server. To determine the buffer size needed, the caller should allocate a buffer in 4096-byte increments until a successful return code is received, or until the buffer size exceeds 65535.

SplEnumQueue Parameter - pReserved

pReserved ([PVOID](#)) - output
Reserved value, must be NULL.

SplEnumQueue Return Value - rc

rc ([SPLERR](#)) - returns
Return code.

- NO_ERROR (0)
No errors occurred.
- ERROR_ACCESS_DENIED (5)
Access is denied.
- ERROR_NOT_SUPPORTED (50)
This request is not supported by the network.
- ERROR_BAD_NETPATH (53)
The network path cannot be located.
- ERROR_INVALID_PARAMETER (87)
An invalid parameter is specified.
- ERROR_INVALID_LEVEL (124)
The level parameter is invalid.
- ERROR_MORE_DATA (234)
Additional data is available.
- NERR_NetNotStarted (2102)
The network program is not started.
- NERR_BufTooSmall (2123)
The API return buffer is too small.
- NERR_InvalidComputer (2351)
The computer name is invalid.

SplEnumQueue - Parameters

pszComputerName (**PSZ**) - input

Name of computer where queues are to be listed.

A NULL string specifies the local workstation.

ulLevel (**ULONG**) - input

Level of detail.

The level of detail required. This must be 3, 4, 5, 6 or 7.

pBuf (**PVOID**) - output

Buffer.

The returned contents in the buffer depend on the value specified in *ulLevel* as follows:

<i>ulLevel</i>	Buffer Contents
3	An array of PRQINFO3 structures. The <i>IsType</i> bit PRQ3_TYPE_APPDEFAULT is set for the application default queue only.
4	An array of <i>*pcReturned</i> PRQINFO3 structures in which each PRQINFO3 structure is followed by an array of PRJINFO2 structures, one for each job in the queue. <i>cJobs</i> in the PRQINFO3 structure gives the number of jobs in the array.
5	An array of PSZ, each pointing to a queue name.
6	An array of PRQINFO6 structures
7	An array of <i>pcReturned</i> PRQINFO3 structures in which each PRQINFO3 structure is followed by an array of PRJINFO4 structures, one for each job in the queue. <i>cJobs</i> in the PRQINFO3 structure gives the number of jobs in the array.

cbBuf (**ULONG**) - input

Size, in bytes, of Buffer.

It must be greater than 0. Some systems do not allow a size greater than 65535.

pcReturned (**PULONG**) - output

Number of entries returned.

pcTotal (**PULONG**) - output

Total number of entries available.

pcbNeeded (**PULONG**) - output

Size in bytes of available information.

A value of 0 specifies that the size is not known. This can occur when enumerating queues on a print server. To determine the buffer size needed, the caller should allocate a buffer in 4096-byte increments until a successful return code is received, or until the buffer size exceeds 65535.

pReserved (**PVOID**) - output

Reserved value, must be NULL.

rc (**SPLERR**) - returns

Return code.

NO_ERROR (0)

No errors occurred.

ERROR_ACCESS_DENIED (5)

Access is denied.

ERROR_NOT_SUPPORTED (50)

This request is not supported by the network.

ERROR_BAD_NETPATH (53)

The network path cannot be located.

ERROR_INVALID_PARAMETER (87)

An invalid parameter is specified.
 ERROR_INVALID_LEVEL (124)
 The level parameter is invalid.
 ERROR_MORE_DATA (234)
 Additional data is available.
 NERR_NetNotStarted (2102)
 The network program is not started.
 NERR_BufTooSmall (2123)
 The API return buffer is too small.
 NERR_InvalidComputer (2351)
 The computer name is invalid.

SplEnumQueue - Related Functions

Related Functions

- [SplQueryJob](#)
 - [SplQueryQueue](#)
 - [SplSetJob](#)
 - [SplSetQueue](#)
-

SplEnumQueue - Example Code

This sample code enumerates all the queues and the jobs in them that are on the local workstation.

```

#define INCL_BASE
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>

INT main ()
{
    SPLERR splerr;
    USHORT jobCount;
    ULONG cbBuf;
    ULONG cTotal;
    ULONG cReturned;
    ULONG cbNeeded;
    ULONG ulLevel;
    ULONG i,j;
    PSZ pszComputerName;
    PBYTE pBuf;
    PPRQINFO3 prq;
    PPRJINFO2 prj2;

    ulLevel = 4L;
    pszComputerName = (PSZ)NULL;
    splerr = SplEnumQueue(pszComputerName,
                          ulLevel,
                          pBuf, 0L, /* cbBuf */
                          &cReturned,
                          &cTotal,
                          &cbNeeded,
                          NULL)
    if (splerr == ERROR_MORE_DATA ||
        splerr == NERR_BufTooSmall)
    {
        if (!DosAllocMem(&pBuf,
                        cbNeeded,
                        PAG_READ |

```

```

        PAG_WRITE |
        PAG_COMMIT))
{
    cbBuf = cbNeeded;
    splerr = SplEnumQueue(pszComputerName,
                          ulLevel,
                          pBuf,
                          cbBuf,
                          &cReturned,
                          &cTotal,
                          &cbNeeded, NULL)

    if (splerr == NO_ERROR)
    {
        /* Set pointer to point to the beginning of the buffer */
        prq = (PPRQINFO3)pBuf;

        /* cReturned has the count of the number of PRQINFO3 structures */
        for (i=0; i < cReturned; i++)
        {
            printf("Queue info: name - %s\n", prq->pszName);
            if (prq->fsType & PRQ3_TYPE_APPDEFAULT)
                printf(" This is the application default print queue\n");
            printf(" priority - %d starttime - %d endtime - %d fsType - %X\n",
                   prq->uPriority,
                   prq->uStartTime,
                   prq->uUntilTime,
                   prq->fsType);
            printf(" pszSepFile - %s\n", prq->pszSepFile);
            printf(" pszPrProc - %s\n", prq->pszPrProc);
            printf(" pszParms - %s\n", prq->pszParms);
            printf(" pszComment - %s\n", prq->pszComment);
            printf(" pszPrinters - %s\n", prq->pszPrinters);
            printf(" pszDriverName- %s\n", prq->pszDriverName);
            if (prq->pDriverData)
            {
                printf(" pDriverData->cb - %ld\n",
                       (ULONG)prq->pDriverData->cb);
                printf(" pDriverData->lVersion - %ld\n",
                       (ULONG)prq->pDriverData->lVersion);
                printf(" pDriverData->szDeviceName- %s\n",
                       prq->pDriverData->szDeviceName);
            }
            /* Save the count of jobs. There are this many PRJINFO2 */
            /* structures following the PRQINFO3 structure */
            jobCount = prq->cJobs;
            printf("Job count in this queue is %d\n\n", jobCount);

            /* Increment the pointer past the PRQINFO3 structure */
            prq++;

            /* Set a pointer to point to the first PRJINFO2 structure */
            prj2=(PPRJINFO2)prq;
            for (j=0; j < jobCount ; j++)
            {
                printf("Job ID = %d\n", prj2->uJobId);
                printf("Job Priority = %d\n", prj2->uPriority);
                printf("User Name = %s\n", prj2->pszUserName);
                printf("Position = %d\n", prj2->uPosition);
                printf("Status = %d\n", prj2->fsStatus);
                printf("Submitted = %ld\n", prj2->ulSubmitted);
                printf("Size = %ld\n", prj2->ulSize);
                printf("Comment = %s\n", prj2->pszComment);
                printf("Document = %s\n\n", prj2->pszDocument);

                /* Increment the pointer to point to the next structure */
                prj2++;
            } /* endfor jobCount */

            /* After doing all the job structures, prj2 points to the next */
            /* queue structure. Set the pointer for a PRQINFO3 structure */
            prq = (PPRQINFO3)prj2;
        } /*endfor cReturned */
    }
    DosFreeMem(pBuf);
}
} /* end if Q level given */
else
{
    /* If we are here we had a bad error code. */

```

```

/* Print it and some other info.          */
printf("SplEnumQueue Error=%ld,
      Total=%ld,
      Returned=%ld,
      Needed=%ld\n",
      splerr,
      cTotal,
      cReturned,
      cbNeeded);
}
DosExit(EXIT_PROCESS, 0);
return(splerr);
} /* end main */

```

SplEnumQueue - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

SplEnumQueueProcessor

SplEnumQueueProcessor - Syntax

This function lists printer queue processors on the local workstation or on a remote server.

```

#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where queues are to be listed. */
ULONG    ulLevel;         /* Level of detail. */
PVOID    pBuf;            /* Buffer. */
ULONG    cbBuf;           /* Size, in bytes, of Buffer. */
PULONG   pcReturned;      /* Number of entries returned. */
PULONG   pcTotal;         /* Total number of entries available. */
PULONG   pcbNeeded;       /* Size in bytes of available information. */
PVOID    pReserved;       /* Reserved value, must be NULL. */
SPLERR   rc;              /* Return code. */

rc = SplEnumQueueProcessor(pszComputerName,
                          ulLevel, pBuf, cbBuf, pcReturned, pcTotal,
                          pcbNeeded, pReserved);

```

SplEnumQueueProcessor Parameter - pszComputerName

pszComputerName (**PSZ**) - input
Name of computer where queues are to be listed.

A NULL string specifies the local workstation.

SplEnumQueueProcessor Parameter - ulLevel

ulLevel (**ULONG**) - input
Level of detail.

The level of detail required. This must be 0.

SplEnumQueueProcessor Parameter - pBuf

pBuf (**PVOID**) - output
Buffer.

The returned contents of the buffer are as follows:

	<i>ulLevel</i>	Buffer Contents
0		An array of PRQPROCINFO structures

SplEnumQueueProcessor Parameter - cbBuf

cbBuf (**ULONG**) - input
Size, in bytes, of Buffer.

It must be greater than 0.

SplEnumQueueProcessor Parameter - pcReturned

pcReturned (**PULONG**) - output
Number of entries returned.

SplEnumQueueProcessor Parameter - pcTotal

pcTotal (**PULONG**) - output
Total number of entries available.

SplEnumQueueProcessor Parameter - pcbNeeded

pcbNeeded (**PULONG**) - output
Size in bytes of available information.

A value of 0 specifies that the size is not known.

SplEnumQueueProcessor Parameter - pReserved

pReserved (**PVOID**) - output
Reserved value, must be NULL.

SplEnumQueueProcessor Return Value - rc

rc (**SPLERR**) - returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
ERROR_MORE_DATA (234)	Additional data is available.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_InvalidComputer (2351)	The computer name is invalid.

SplEnumQueueProcessor - Parameters

pszComputerName (**PSZ**) - input
Name of computer where queues are to be listed.

A NULL string specifies the local workstation.

ulLevel (**ULONG**) - input
Level of detail.

The level of detail required. This must be 0.

pBuf (**PVOID**) - output
Buffer.

The returned contents of the buffer are as follows:

<i>ulLevel</i>	Buffer Contents
0	An array of PRQPROCINFO structures

cbBuf (**ULONG**) - input
Size, in bytes, of Buffer.

It must be greater than 0.

pcReturned (**PULONG**) - output
Number of entries returned.

pcTotal (**PULONG**) - output
Total number of entries available.

pcbNeeded (**PULONG**) - output
Size in bytes of available information.

A value of 0 specifies that the size is not known.

pReserved (**PVOID**) - output
Reserved value, must be NULL.

rc (**SPLERR**) - returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
ERROR_MORE_DATA (234)	Additional data is available.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_InvalidComputer (2351)	The computer name is invalid.

SplEnumQueueProcessor - Related Functions

Related Functions

- [SplSetQueue](#)

SplEnumQueueProcessor - Example Code

This sample code enumerates and prints all the queue processors on the local computer.

```
#define INCL_BASE
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>          /* for printf function    */

INT main ()
{
    SPLERR splerr ;
    ULONG  cbBuf ;
    ULONG  cTotal ;
    ULONG  cReturned ;
    ULONG  cbNeeded ;
    ULONG  i ;
    PSZ    pszComputerName = NULL ;
    PSZ    pszQProcName ;
    PBYTE  pBuf ;

    /* Call the function the first time with zero in cbBuf. The count
    /* of bytes needed for the buffer to hold all the info will be
    /* returned in cbNeeded.
    splerr = SplEnumQueueProcessor(pszComputerName, 0L, NULL, 0L,
                                   &cReturned, &cTotal,
                                   &cbNeeded, NULL );

    /* If the return code is ERROR_MORE_DATA or NERR_BufTooSmall,
    /* then all the parameters were correct; and we can continue.
    if (splerr == ERROR_MORE_DATA || splerr == NERR_BufTooSmall)
    {
        /* Allocate memory for the buffer to hold the returned information. Use
        /* the count of bytes that were returned by our first call.
        if (!DosAllocMem( &pBuf, cbNeeded,
                        PAG_READ|PAG_WRITE|PAG_COMMIT) )
        {
            /* Set count of bytes to the value returned by our first call.
            cbBuf = cbNeeded ;

            /* Now call the function a second time with the correct values, and
            /* the information will be returned in the buffer.
            splerr = SplEnumQueueProcessor(pszComputerName, 0L, pBuf, cbBuf,
                                           &cReturned, &cTotal,
                                           &cbNeeded, NULL );

            /* If we have no errors, then print out the buffer information.
            if (splerr == NO_ERROR)
            {
                /* Set a pointer to point to the beginning of the buffer.
                pszQProcName = (PSZ)pBuf;

                /* Print the names that are in the buffer. The count of the number
                /* of names in pBuf have been returned in cReturned.
                for (i=0; i < cReturned ; i++)
                {
                    printf("Queue Processor name - %s\n", pszQProcName) ;

                    /* Increment the pointer to point to the next name.
                    pszQProcName += DRV_NAME_SIZE + 1;
                }
            }
            /* Free the memory allocated for the buffer.
            DosFreeMem(pBuf) ;
        }
    }
```

```

    }
    else
    {
        /* If the first call to the function returned any other error code */
        /* except ERROR_MORE_DATA or NERR_BufTooSmall, we print the */
        /* following. */
        printf("SplEnumQueueProcessor error=%ld\n", splerr ) ;
    }
    DosExit( EXIT_PROCESS , 0 ) ;
    return (splerr);
}

```

SplEnumQueueProcessor - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

SplGetControlPanelList

SplGetControlPanelList - Syntax

Description:

This function queries installed control panel DLLs for support for the printer attached to the given port.

```

#define INCL_SPL
#define INCL_SPLBIDI
#include <os2.h>

PSZ      pszComputerName; /* Name of computer connected to the printer port. */
PSZ      pszPortName;    /* Name of printer port on pszComputerName used to determine if a control panel ca
PSZ      pszDeviceID;    /* Device ID, if known, for the printer connected to pszPortName. */
ULONG    flCapabilities; /* Compatibilities requested. */
PSZ      pszListOfPanels; /* Receives comma-separated list of control panel names capable of displaying a co
ULONG    cbBuf;          /* Length of buffer, in bytes, pointed to by pszListOfPanels. */
PULONG   pcbNeeded;      /* Receives length, in bytes, of buffer needed to store entire list of control pan
ULONG    rc;              /* Return codes. */

rc = SplGetControlPanelList(pszComputerName,
    pszPortName, pszDeviceID, flCapabilities,
    pszListOfPanels, cbBuf, pcbNeeded);

```

SplGetControlPanelList Parameter - pszComputerName

pszComputerName (PSZ) - input
Name of computer connected to the printer port.

This is NULL for printers whose port exists on the current machine.

SplGetControlPanelList Parameter - pszPortName

pszPortName (PSZ) - input
Name of printer port on *pszComputerName* used to determine if a control panel can be displayed.

SplGetControlPanelList Parameter - pszDeviceID

pszDeviceID (PSZ) - input
Device ID, if known, for the printer connected to *pszPortName*.

SplGetControlPanelList Parameter - flCapabilities

flCapabilities (ULONG) - input
Compatibilities requested.

FL_ADMIN - 0x00000001 Set if an administrator wants the control panel.

SplGetControlPanelList Parameter - pszListOfPanels

pszListOfPanels (PSZ) - output
Receives comma-separated list of control panel names capable of displaying a control panel for the printer.

SplGetControlPanelList Parameter - cbBuf

cbBuf (ULONG) - input
Length of buffer, in bytes, pointed to by *pszListOfPanels*.

SplGetControlPanelList Parameter - pcbNeeded

pcbNeeded (**PULONG**) - output

Receives length, in bytes, of buffer needed to store entire list of control panels for the printer.

SplGetControlPanelList Return Value - rc

rc (**ULONG**) - returns

Return codes.

0 Success

ERROR_NOT_SUPPORTED(50)

A control panel for this printer is not supported by any control panel DLL.

NERR_BufTooSmall(2123)

Need to call again with a bigger buffer.

SplGetControlPanelList - Parameters

pszComputerName (**PSZ**) - input

Name of computer connected to the printer port.

This is NULL for printers whose port exists on the current machine.

pszPortName (**PSZ**) - input

Name of printer port on *pszComputerName* used to determine if a control panel can be displayed.

pszDeviceID (**PSZ**) - input

Device ID, if known, for the printer connected to *pszPortName*.

flCapabilities (**ULONG**) - input

Compatibilities requested.

FL_ADMIN - 0x00000001

Set if an administrator wants the control panel.

pszListOfPanels (**PSZ**) - output

Receives comma-separated list of control panel names capable of displaying a control panel for the printer.

cbBuf (**ULONG**) - input

Length of buffer, in bytes, pointed to by *pszListOfPanels*.

pcbNeeded (**PULONG**) - output

Receives length, in bytes, of buffer needed to store entire list of control panels for the printer.

rc (**ULONG**) - returns

Return codes.

0	Success
ERROR_NOT_SUPPORTED(50)	A control panel for this printer is not supported by any control panel DLL.
NERR_BufTooSmall(2123)	Need to call again with a bigger buffer.

SplGetControlPanelList - Remarks

This function will return Success only for printers that are configured to communicate bidirectionally.

SplGetControlPanelList - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

SplGetPortFromQ

SplGetPortFromQ - Syntax

Description:

This function gets a list of ports connected to the print queue for the print object or applications. This function is used by the print object to determine if some menu items can be enabled based on the type of printer port connected to a print queue.

```
#define INCL_SPL
#define INCL_SPLBIDI
#include <os2.h>

PSZ      pszComputerName; /* Name of the computer on which pszQueueName exists. */
PSZ      pszQueueName;   /* Name of print queue. */
ULONG    ulLevel;        /* Level of information to return. */
PVOID    pBuf;           /* Receives the GETPORTFROMQ data structure. */
ULONG    cbBuf;          /* Length of buffer pointed to by pBuf, in bytes. */
ULONG    pcbNeeded;       /* Receives length, in bytes, of buffer needed to return all port data. */
ULONG    rc;              /* Return codes. */

rc = SplGetPortFromQ(pszComputerName, pszQueueName,
                    ulLevel, pBuf, cbBuf, pcbNeeded);
```

SplGetPortFromQ Parameter - pszComputerName

pszComputerName (**PSZ**) - input

Name of the computer on which *pszQueueName* exists.

This value is NULL if the queue is local or if the queue references a network print queue.

SplGetPortFromQ Parameter - pszQueueName

pszQueueName (**PSZ**) - input

Name of print queue.

This is the print queue name on *pszComputerName* or a local print queue name that may reference a network print queue.

SplGetPortFromQ Parameter - ulLevel

ulLevel (**ULONG**) - input

Level of information to return.

Only level 1 is supported, which returns the GETPORTFROMQ data structure.

SplGetPortFromQ Parameter - pBuf

pBuf (**PVOID**) - output

Receives the GETPORTFROMQ data structure.

SplGetPortFromQ Parameter - cbBuf

cbBuf (**ULONG**) - input

Length of buffer pointed to by *pBuf*, in bytes.

SplGetPortFromQ Parameter - pcbNeeded

pcbNeeded (ULONG) - output
Receives length, in bytes, of buffer needed to return all port data.

SplGetPortFromQ Return Value - rc

rc (ULONG) - returns
Return codes.

0	Success
ERROR_BAD_NET_NAME(67)	The computer name cannot be found.
ERROR_BAD_NETPATH(53)	The computer name specified is not available or is invalid.
ERROR_INVALID_LEVEL(124)	<i>ulLevel</i> is not 1.
ERROR_INVALID_PARAMETER(87)	An invalid parameter was specified; most likely an invalid <i>pBuf</i> value.
ERROR_MORE_DATA(234)	Not all data could fit in <i>pBuf</i> .
ERROR_NOT_SUPPORTED(50)	Command not supported on <i>pszComputerName</i> .
NERR_BufTooSmall(2123)	No data could fit in <i>pBuf</i> .
NERR_QNotFound(2150)	The queue name does not exist.

SplGetPortFromQ - Parameters

pszComputerName (PSZ) - input
Name of the computer on which *pszQueueName* exists.

This value is NULL if the queue is local or if the queue references a network print queue.

pszQueueName (PSZ) - input
Name of print queue.

This is the print queue name on *pszComputerName* or a local print queue name that may reference a network print queue.

ulLevel (ULONG) - input
Level of information to return.

Only level 1 is supported, which returns the GETPORTFROMQ data structure.

pBuf (PVOID) - output
Receives the GETPORTFROMQ data structure.

cbBuf ([ULONG](#)) - input
Length of buffer pointed to by *pBuf*, in bytes.

pcbNeeded ([ULONG](#)) - output
Receives length, in bytes, of buffer needed to return all port data.

rc ([ULONG](#)) - returns
Return codes.

0	Success
ERROR_BAD_NET_NAME(67)	The computer name cannot be found.
ERROR_BAD_NETPATH(53)	The computer name specified is not available or is invalid.
ERROR_INVALID_LEVEL(124)	<i>ulLevel</i> is not 1.
ERROR_INVALID_PARAMETER(87)	An invalid parameter was specified; most likely an invalid <i>pBuf</i> value.
ERROR_MORE_DATA(234)	Not all data could fit in <i>pBuf</i> .
ERROR_NOT_SUPPORTED(50)	Command not supported on <i>pszComputerName</i> .
NERR_BufTooSmall(2123)	No data could fit in <i>pBuf</i> .
NERR_QNotFound(2150)	The queue name does not exist.

SplGetPortFromQ - Remarks

None.

SplGetPortFromQ - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Glossary](#)

SplHoldJob

SplHoldJob - Syntax

This function holds a job in a print queue.

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where job is to be paused. */
PSZ      pszQueueName;    /* Queue Name. */
ULONG    ulJob;           /* Job identification number. */
SPLERR   rc;              /* Return code. */

rc = SplHoldJob(pszComputerName, pszQueueName,
               ulJob);
```

SplHoldJob Parameter - pszComputerName

pszComputerName (**PSZ**) - input
Name of computer where job is to be paused.

A NULL string specifies the local workstation.

SplHoldJob Parameter - pszQueueName

pszQueueName (**PSZ**) - input
Queue Name.

SplHoldJob Parameter - ulJob

ulJob (**ULONG**) - input
Job identification number.

SplHoldJob Return Value - rc

rc (**SPLERR**) - returns
Return code.

NO_ERROR (0)
No errors occurred.

ERROR_ACCESS_DENIED (5)
Access is denied.
ERROR_NOT_SUPPORTED (50)
This request is not supported by the network.
ERROR_BAD_NETPATH (53)
The network path cannot be located.
NERR_NetNotStarted (2102)
The network program is not installed.
NERR_JobNotFound (2151)
The print job does not exist.
NERR_SpoolerNotLoaded (2161)
The spooler is not running.
NERR_JobInvalidState (2164)
This operation cannot be performed on the print job in its current state.
NERR_InvalidComputer (2351)
The computer name is invalid.

SplHoldJob - Parameters

pszComputerName ([PSZ](#)) - input
Name of computer where job is to be paused.

A NULL string specifies the local workstation.

pszQueueName ([PSZ](#)) - input
Queue Name.

ulJob ([ULONG](#)) - input
Job identification number.

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)
No errors occurred.
ERROR_ACCESS_DENIED (5)
Access is denied.
ERROR_NOT_SUPPORTED (50)
This request is not supported by the network.
ERROR_BAD_NETPATH (53)
The network path cannot be located.
NERR_NetNotStarted (2102)
The network program is not installed.
NERR_JobNotFound (2151)
The print job does not exist.
NERR_SpoolerNotLoaded (2161)
The spooler is not running.
NERR_JobInvalidState (2164)
This operation cannot be performed on the print job in its current state.
NERR_InvalidComputer (2351)
The computer name is invalid.

SplHoldJob - Remarks

If the job is already printing when the call is made, NERR_JobInvalidState (2164) is returned.

A user with administrator privilege can hold any job.

A job created locally can be held locally regardless of user privilege level, but can be held remotely only by an administrator.

A remote job can be held by a user without administrator privilege only if the username of the person initiating the request is the same as the username of the person who created the job.

SplHoldJob - Related Functions

Related Functions

- [SplEnumJob](#)
 - [SplQueryJob](#)
 - [SplReleaseJob](#)
-

SplHoldJob - Example Code

This sample code will hold the queue name that is entered at the prompt.

```
#define INCL_BASE
#define INCL_SPL
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>          /* for printf function */
#include <stdlib.h>         /* for atoi function  */

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    SPLERR splerr ;
    PSZ    pszComputerName = NULL ;
    PSZ    pszQueueName = NULL ;
    ULONG  ulJob ;

    /* Get job id from the input argument.  */
    ulJob = atoi(argv[1]);

    /* Call the function to do the hold. If an error is returned, print it. */
    splerr = SplHoldJob( pszComputerName, pszQueueName, ulJob);

    switch (splerr)
    {
        case NO_ERROR:
            printf("Job %d was held.\n",ulJob);
            break;
        case NERR_JobNotFound:
            printf("Job does not exist.\n");
            break;
        case NERR_JobInvalidState:
            printf("This operation can't be performed on the print Job.\n");
            break;
        default:
            printf("Errorcode = %ld\n",splerr);
    } /* endswitch */
    DosExit( EXIT_PROCESS , 0 ) ;
    argc;
    return (splerr);
}
```

SplHoldJob - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

SplHoldQueue

SplHoldQueue - Syntax

This function holds a print queue.

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where queue is to be paused. */
PSZ      pszQueueName;   /* Queue name. */
SPLERR   rc;             /* Return code. */

rc = SplHoldQueue(pszComputerName, pszQueueName);
```

SplHoldQueue Parameter - pszComputerName

pszComputerName ([PSZ](#)) - input

Name of computer where queue is to be paused.

A NULL string specifies the local workstation.

SplHoldQueue Parameter - pszQueueName

pszQueueName ([PSZ](#)) - input

Queue name.

SplHoldQueue Return Value - rc

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_QNotFound (2150)	The printer queue does not exist.
NERR_InvalidComputer (2351)	The computer name is invalid.

SplHoldQueue - Parameters

pszComputerName ([PSZ](#)) - input
Name of computer where queue is to be paused.

A NULL string specifies the local workstation.

pszQueueName ([PSZ](#)) - input
Queue name.

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_QNotFound (2150)	The printer queue does not exist.
NERR_InvalidComputer (2351)	The computer name is invalid.

SplHoldQueue - Remarks

This function suspends processing of all print jobs except for a job currently printing. Print jobs can be submitted to a held queue, but no jobs will be spooled to a print destination or print processor until the queue is released by a SplHoldQueue call.

To hold a remote queue requires administrator privilege on the remote server.

SplHoldQueue - Related Functions

Related Functions

- [SplCreateQueue](#)
 - [SplEnumQueue](#)
 - [SplQueryQueue](#)
 - [SplReleaseQueue](#)
-

SplHoldQueue - Example Code

This sample code will hold the local queue name that is entered at the prompt.

```
#define INCL_SPL
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>          /* for printf function */

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    SPLERR splerr ;
    PSZ    pszComputerName = NULL ;
    PSZ    pszQueueName ;

    /* Get queue name from the input argument */
    pszQueueName = argv[1];

    /* Call the function to do the hold. If an error is returned, print it. */
    splerr = SplHoldQueue(pszComputerName, pszQueueName);
    if (splerr != 0L)
    {
        switch (splerr)
        {
            case NERR_QNotFound:
                printf("Queue does not exist.\n");
                break;
            case NERR_SpoolerNotLoaded:
                printf("The Spooler is not running.\n");
                break;
            default:
                printf("Errorcode = %ld\n",splerr);
        } /* endswitch */
    }
    else
    {
        printf("Queue %s was held.\n",pszQueueName);
    } /* endif */
    DosExit( EXIT_PROCESS , 0 ) ;
    argc; /* keep the compiler quiet */
    return (splerr);
}
```

SplHoldQueue - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

SplMessageBox

SplMessageBox - Syntax

Description:

SplMessageBox creates and displays a message box.

The message queue will be created (and destroyed) if necessary. In OS/2 2.1, this API may return Retry automatically if the PM message box is displayed for more than three minutes.

```
#include <os2.h>

PSZ      pszAddress; /* Pointer to a string containing the logical address of the device, such as LPT1. */
ULONG    flErrorInfo; /* Error information. */
ULONG    flErrorData; /* Length of the data in bytes. */
PSZ      pszText; /* Pointer to the text string for the message box. */
PSZ      pszCaption; /* Pointer to a string containing a meaningful title for the message box. */
ULONG    idWindow; /* Window ID of the message box window. */
ULONG    fsStyle; /* Bit array specifying the contents and function of the message box. */
ULONG    Response; /* Return codes. */

Response = SplMessageBox(pszAddress, flErrorInfo,
                        flErrorData, pszText, pszCaption,
                        idWindow, fsStyle);
```

SplMessageBox Parameter - pszAddress

pszAddress ([PSZ](#)) - input

Pointer to a string containing the logical address of the device, such as LPT1.

SplMessageBox Parameter - flErrorInfo

flErrorInfo ([ULONG](#)) - input

Error information.

One of the following flags must be set to identify where the error occurred:

SPLINFO_QPERROR	Spooler queue processor error
SPLINFO_DDERROR	Presentation driver error
SPLINFO_SPLERROR	Spooler error
SPLINFO_OTHERERROR	Any other error

One of the following flags is also set to indicate the severity of the error:

SPLINFO_INFORMATION	Information only, no error.
SPLINFO_WARNING	Warning.
SPLINFO_ERROR	Recoverable error.
SPLINFO_SEVERE	Severe, unrecoverable error.
SPLINFO_USERINTREQD	This flag is optional. It shows that recovery requires action from the user.

SplMessageBox Parameter - flErrorData

flErrorData ([ULONG](#)) - input
Length of the data in bytes.

Error data:

SPLDATA_PRINTERJAM	Printer is jammed, offline, or not turned on
SPLDATA_FORMCHGREQD	Form change required
SPLDATA_CARTCHGREQD	Font cartridge change required
SPLDATA_PENCHGREQD	Pen change required
SPLDATA_DATAERROR	Data error, such as missing file
SPLDATA_UNEXPECTERROR	Unexpected DOS error
SPLDATA_OTHER	Any other error

SplMessageBox Parameter - pszText

pszText ([PSZ](#)) - input
Pointer to the text string for the message box.

SplMessageBox Parameter - pszCaption

pszCaption ([PSZ](#)) - input
Pointer to a string containing a meaningful title for the message box.

The text is centered in the title bar. If more than 40 characters are supplied, excess characters at the beginning and end of the string are not displayed.

SplMessageBox Parameter - idWindow

idWindow (ULONG) - input
Window ID of the message box window.

SplMessageBox Parameter - fsStyle

fsStyle (ULONG) - input
Bit array specifying the contents and function of the message box.

SplMessageBox Return Value - Response

Response (ULONG) - returns
Return codes.

This function returns a ULONG value (sResponse) that indicates the user's response.

MBID_ENTER	ENTER push button was selected
MBID_OK	OK push button was selected
MBID_CANCEL	CANCEL push button was selected
MBID_ABORT	ABORT push button was selected
MBID_RETRY	RETRY push button was selected
MBID_IGNORE	IGNORE push button was selected
MBID_YES	YES push button was selected
MBID_NO	NO push button was selected
MBID_ERROR	Function not successful; an error occurred.

SplMessageBox - Parameters

pszAddress (PSZ) - input
Pointer to a string containing the logical address of the device, such as LPT1.

flErrorInfo (ULONG) - input
Error information.

One of the following flags must be set to identify where the error occurred:

SPLINFO_QPERROR	Spooler queue processor error
SPLINFO_DDERROR	Presentation driver error
SPLINFO_SPLERROR	Spooler error
SPLINFO_OTHERERROR	Any other error

One of the following flags is also set to indicate the severity of the error:

SPLINFO_INFORMATION	Information only, no error.
SPLINFO_WARNING	Warning.
SPLINFO_ERROR	Recoverable error.
SPLINFO_SEVERE	Severe, unrecoverable error.
SPLINFO_USERINTREQD	This flag is optional. It shows that recovery requires action from the user.

flErrorData (ULONG) - input
Length of the data in bytes.

Error data:

SPLDATA_PRINTERJAM	Printer is jammed, offline, or not turned on
SPLDATA_FORMCHGREQD	Form change required
SPLDATA_CARTCHGREQD	Font cartridge change required
SPLDATA_PENCHGREQD	Pen change required
SPLDATA_DATAERROR	Data error, such as missing file
SPLDATA_UNEXPECTERROR	Unexpected DOS error
SPLDATA_OTHER	Any other error

pszText (PSZ) - input
Pointer to the text string for the message box.

pszCaption (PSZ) - input
Pointer to a string containing a meaningful title for the message box.

The text is centered in the title bar. If more than 40 characters are supplied, excess characters at the beginning and end of the string are not displayed.

idWindow (ULONG) - input
Window ID of the message box window.

fsStyle (ULONG) - input
Bit array specifying the contents and function of the message box.

Response (ULONG) - returns
Return codes.

This function returns a ULONG value (sResponse) that indicates the user's response.

MBID_ENTER	ENTER push button was selected
MBID_OK	OK push button was selected
MBID_CANCEL	CANCEL push button was selected
MBID_ABORT	ABORT push button was selected
MBID_RETRY	RETRY push button was selected
MBID_IGNORE	IGNORE push button was selected
MBID_YES	YES push button was selected
MBID_NO	NO push button was selected
MBID_ERROR	Function not successful; an error occurred.

SplMessageBox - Remarks

SplMessageBox creates a message queue if the current thread does not have one.

SplMessageBox is similar to [WinMessageBox](#).

SplMessageBox - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

SplPurgeQueue

SplPurgeQueue - Syntax

This function removes all jobs, except any currently printing, from a print queue.

```
#define INCL_SPL /* Or use INCL_PM, */  
#include <os2.h>  
  
PSZ      pszComputerName; /* Name of computer where queue is to be purged. */  
PSZ      pszQueueName;   /* Queue name. */  
SPLERR   rc;             /* Return code. */  
  
rc = SplPurgeQueue(pszComputerName, pszQueueName);
```

SplPurgeQueue Parameter - pszComputerName

pszComputerName ([PSZ](#)) - input
Name of computer where queue is to be purged.

A NULL string specifies the local workstation.

SplPurgeQueue Parameter - pszQueueName

pszQueueName ([PSZ](#)) - input
Queue name.

SplPurgeQueue Return Value - rc

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)
No errors occurred.
ERROR_ACCESS_DENIED (5)
Access is denied.
ERROR_NOT_SUPPORTED (50)
This request is not supported by the network.
ERROR_BAD_NETPATH (53)
The network path cannot be located.
ERROR_INVALID_PARAMETER (87)
An invalid parameter was specified.
NERR_NetNotStarted (2102)
The network program is not started.
NERR_QNotFound (2150)
The printer queue does not exist.
NERR_SpoolerNotLoaded (2161)
The spooler is not running.
NERR_InvalidComputer (2351)
The computer name is invalid.

SplPurgeQueue - Parameters

pszComputerName ([PSZ](#)) - input
Name of computer where queue is to be purged.

A NULL string specifies the local workstation.

pszQueueName ([PSZ](#)) - input
Queue name.

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)
No errors occurred.
ERROR_ACCESS_DENIED (5)
Access is denied.
ERROR_NOT_SUPPORTED (50)
This request is not supported by the network.
ERROR_BAD_NETPATH (53)
The network path cannot be located.
ERROR_INVALID_PARAMETER (87)
An invalid parameter was specified.
NERR_NetNotStarted (2102)
The network program is not started.
NERR_QNotFound (2150)
The printer queue does not exist.
NERR_SpoolerNotLoaded (2161)
The spooler is not running.
NERR_InvalidComputer (2351)
The computer name is invalid.

SplPurgeQueue - Remarks

A print job that is printing is not affected by this function.

If a print queue is pending deletion when this function is made, the queue is deleted when the job that is currently printing ends.

To purge a remote queue requires administrator privilege on the remote server.

SplPurgeQueue - Related Functions

Related Functions

- [SplCreateQueue](#)
 - [SplEnumQueue](#)
 - [SplQueryQueue](#)
-

SplPurgeQueue - Example Code

This code will purge a local queue, whose name is entered at the prompt.

```
#define INCL_SPL
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>          /* for printf function */

INT main (argc, argv)
    INT argc;
    CHAR *arg[];
{
    SPLERR splerr ;
    PSZ    pszComputerName = NULL ;
    PSZ    pszQueueName ;

    /* Get queue name from the input argument. */
    pszQueueName = arg[1];

    /* Call the function to do the purge. If an error is returned, print it. */
    splerr=SplPurgeQueue(pszComputerName, pszQueueName);
    if (splerr != 0L)
    {
        switch (splerr)
        {
            case NERR_QNotFound:
                printf("Queue does not exist.\n");
                break;
            case NERR_SpoolerNotLoaded:
                printf("The Spooler is not running.\n");
                break;
            default:
                printf("Errorcode = %ld\n",splerr);
        } /* endswitch */
    }
    else
    {
        printf("Queue %s was purged.\n",pszQueueName);
    } /* endif */

    DosExit( EXIT_PROCESS , 0 ) ;
    return (splerr);
}
```

SplPurgeQueue - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

SplQmAbort

SplQmAbort - Syntax

Description:

SplQmAbort stops the generation of the spool file. It automatically closes the spool file (see [SplQmClose](#)).

```
#define INCL_SPL /* Or use INCL_PM, */  
#include <os2.h>  
  
HSPL    hspl; /* Spooler handle. */  
BOOL    rc;   /* Success indicator. */  
  
rc = SplQmAbort(hspl);
```

SplQmAbort Parameter - hspl

hspl ([HSPL](#)) - input
Spooler handle.

SplQmAbort Return Value - rc

rc ([BOOL](#)) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

SplQmAbort - Parameters

hspl ([HSPL](#)) - input
Spooler handle.

rc ([BOOL](#)) - returns
Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

SplQmAbort - Errors

Possible returns from [WinGetLastError](#)

PMERR_SPL_QUEUE_ERROR (0x4004)
No spooler queue supplied or found.

PMERR_SPL_INV_HSPL (0x4005)
The spooler handle is invalid.

SplQmAbort - Related Functions

Prerequisite Functions

- [SplQmOpen](#)

Related Functions

- [DevEscape](#)
-

SplQmAbort - Example Code

This function is used to stop the generation of spool files and automatically close the spool file.

```
#define INCL_SPL
#include <OS2.H>
```

```
HSPL hspl; /* spooler handle. */  
SplQmAbort(hspl);
```

SplQmAbort - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

SplQmAbortDoc

SplQmAbortDoc - Syntax

Description:

SplQmAbortDoc ends a print job.

```
#define INCL_SPL /* Or use INCL_PM, */  
#include <os2.h>  
  
HSPL    hspl; /* Spooler handle. */  
BOOL    rc;   /* Success indicator. */  
  
rc = SplQmAbortDoc(hspl);
```

SplQmAbortDoc Parameter - hspl

hspl ([HSPL](#)) - input
Spooler handle.

SplQmAbortDoc Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SplQmAbortDoc - Parameters

hspl ([HSPL](#)) - input
Spooler handle.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SplQmAbortDoc - Errors

Possible returns from [WinGetLastError](#)

PMERR_SPL_QUEUE_ERROR (0x4004)
No spooler queue supplied or found.

PMERR_STARTDOC_NOT_ISSUED (0x2104)
A request to write spooled output without first issuing a STARTDOC was attempted.

PMERR_SPL_INV_HSPL (0x4005)
The spooler handle is invalid.

SplQmAbortDoc - Related Functions

Prerequisite Functions

- [SplQmOpen](#)
- [SplQmStartDoc](#)

Related Functions

- [DevEscape](#)

SplQmAbortDoc - Example Code

This function is used to end a print job. Everything that has been written to the spool file for this job since the last SplQmStartDoc is erased, including the SplQmStartDoc.

```
#define INCL_SPL
#include <OS2.H>

HSPL hspl; /* spooler handle. */

SplQmAbortDoc(hspl);
```

SplQmAbortDoc - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

SplQmClose

SplQmClose - Syntax

Description:

SplQmClose closes the spool file. It corresponds to the [DevCloseDC](#) function.

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

HSPL    hspl; /* Spooler handle. */
BOOL    rc;    /* Success indicator. */

rc = SplQmClose(hspl);
```

SplQmClose Parameter - hspl

hspl ([HSPL](#)) - input
Spooler handle.

SplQmClose Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SplQmClose - Parameters

hspl ([HSPL](#)) - input
Spooler handle.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SplQmClose - Errors

Possible returns from [WinGetLastError](#)

PMERR_SPL_QUEUE_ERROR (0x4004)
No spooler queue supplied or found.

PMERR_ENDDOC_NOT_ISSUED (0x210B)
A request to close the spooled output without first issuing a an ENDDOC was attempted.

PMERR_SPL_INV_HSPL (0x4005)
The spooler handle is invalid.

SplQmClose - Related Functions

- Prerequisite Functions**
- [SplQmOpen](#)

Related Functions

- [DevCloseDC](#)

SplQmClose - Example Code

This function is used to close a spool file that was opened with [SplQmOpen](#).

```
#define INCL_SPL
#include <OS2.H>

HSPL hspl; /* spooler handle. */

SplQmClose(hspl);
```

SplQmClose - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

SplQmEndDoc

SplQmEndDoc - Syntax

Description:

SplQmEndDoc ends a print job, and returns ulJob, a unique number to identify the job. This function corresponds to the [DevEscape](#) (DEVESC_ENDDOC) call.

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

HSPL    hspl;    /* Spooler handle. */
ULONG   ulJob;   /* Job identifier. */

ulJob = SplQmEndDoc(hspl);
```

SplQmEndDoc Parameter - hspl

hspl ([HSPL](#)) - input
Spooler handle.

SplQmEndDoc Return Value - ulJob

ulJob ([ULONG](#)) - returns
Job identifier.

Nonzero	Jobid (1 through 65,535)
SPL_ERROR	Error

SplQmEndDoc - Parameters

hspl ([HSPL](#)) - input
Spooler handle.

ulJob ([ULONG](#)) - returns
Job identifier.

Nonzero	Jobid (1 through 65,535)
SPL_ERROR	Error

SplQmEndDoc - Errors

Possible returns from [WinGetLastError](#)

PMERR_SPL_QUEUE_ERROR (0x4004)
No spooler queue supplied or found.

PMERR_SPL_NO_DATA (0x400D)
No data supplied or found.

PMERR_SPL_INV_HSPL (0x4005)
The spooler handle is invalid.

SplQmEndDoc - Related Functions

Prerequisite Functions

- [SplQmOpen](#)
- [SplQmStartDoc](#)

Related Functions

- [DevEscape](#)

SplQmEndDoc - Example Code

This function is used to end a print job and return the job id. The print-job identifier is displayed to the user by the spooler while this job is on the queue, and while it is being printed.

```
#define INCL_SPL
#include <OS2.H>

HSPL hspl; /* spooler handle. */
ULONG jobid;
CHAR szMsg[100];
HWND hwndClient;

jobid = SplQmEndDoc(hspl);

sprintf(szMsg, "ending job %d", jobid);
WinMessageBox(HWND_DESKTOP,
    hwndClient,          /* client-window handle */
    szMsg,               /* body of the message */
    "Printing Information", /* title of the message */
    0,                   /* message box id */
    MB_NOICON | MB_OK); /* icon and button flags */
```

SplQmEndDoc - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

SplQmGetJobID

SplQmGetJobID - Syntax

Description:

This function returns the spooler print job ID for the spool file handle.

This function allows Presentation drivers (or applications) to get the spooler print job ID while still spooling the job. Currently, only [SplQmEndDoc](#) will return the spooler print job ID.

SplQmGetJobID will return an error if called after [SplQmEndDoc](#) and before the next [SplQmStartDoc](#) because the spool file handle does not reference a spool job after SplQmEndDoc.

```
#define INCL_SPL
#define INCL_SPLBIDI
#include <os2.h>

HSPL      hspl;          /* Spool file handle returned by SplQmOpen. */
ULONG     ulLevel;       /* Level of job information; currently, level 0 is supported. */
PVOID     pBuf;          /* Buffer to receive job information. */
ULONG     cbBuf;         /* Length of pBuf, in bytes. */
PULONG    pcbNeeded;     /* Receives length, in bytes, of buffer needed to store job information. */
ULONG     rc;            /* Return codes. */

rc = SplQmGetJobID(hspl, ulLevel, pBuf, cbBuf,
                  pcbNeeded);
```

SplQmGetJobID Parameter - hspl

hspl ([HSPL](#)) - input
Spool file handle returned by SplQmOpen.

SplQmGetJobID Parameter - ulLevel

ulLevel ([ULONG](#)) - input
Level of job information; currently, level 0 is supported.

SplQmGetJobID Parameter - pBuf

pBuf ([PVOID](#)) - output
Buffer to receive job information.

Level 0 will return the QMJOBINFO data structure.

SplQmGetJobID Parameter - cbBuf

cbBuf ([ULONG](#)) - input
Length of *pBuf*, in bytes.

SplQmGetJobID Parameter - pcbNeeded

pcbNeeded ([PULONG](#)) - output
Receives length, in bytes, of buffer needed to store job information.

SplQmGetJobID Return Value - rc

rc ([ULONG](#)) - returns
Return codes.

0	Success
ERROR_INVALID_HANDLE(6)	Invalid handle given.
ERROR_INVALID_PARAMETER(87)	<i>ulLevel</i> not 0 or invalid parameter given.
ERROR_MORE_DATA(234)	Part of the information requested was returned, but there was not enough room for all the data. The <i>*pcbOutData</i> parameter contains the size of buffer needed to retrieve all the query response data.
ERROR_NOT_SUPPORTED(50)	This function is not supported by the print server processing the spool job.
NERR_BufTooSmall(2123)	No information was returned because the output buffer was too small. <i>*pcbOutData</i> contains the size of buffer needed to retrieve all the query response data.
NERR_JobNotFound(2151)	No spooler job found.

SplQmGetJobID - Parameters

hspl ([HSPL](#)) - input
Spool file handle returned by SplQmOpen.

ulLevel ([ULONG](#)) - input
Level of job information; currently, level 0 is supported.

pBuf ([PVOID](#)) - output
Buffer to receive job information.

Level 0 will return the QMJOBINFO data structure.

cbBuf ([ULONG](#)) - input
Length of *pBuf*, in bytes.

pcbNeeded ([PULONG](#)) - output
Receives length, in bytes, of buffer needed to store job information.

rc ([ULONG](#)) - returns
Return codes.

0	Success
ERROR_INVALID_HANDLE(6)	Invalid handle given.
ERROR_INVALID_PARAMETER(87)	<i>uiLevel</i> not 0 or invalid parameter given.
ERROR_MORE_DATA(234)	Part of the information requested was returned, but there was not enough room for all the data. The <i>*pcbOutData</i> parameter contains the size of buffer needed to retrieve all the query response data.
ERROR_NOT_SUPPORTED(50)	This function is not supported by the print server processing the spool job.
NERR_BufTooSmall(2123)	No information was returned because the output buffer was too small. <i>*pcbOutData</i> contains the size of buffer needed to retrieve all the query response data.
NERR_JobNotFound(2151)	No spooler job found.

SplQmGetJobID - Remarks

None.

SplQmGetJobID - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Glossary](#)

SplQmNewPage

SplQmNewPage - Syntax

Description:

This function notifies the spooler of a page boundary when spooling a print job. It is called by the Presentation driver just prior to the first [SplQmWrite](#) for a page, including before the first page of the print job being spooled.

SplQmNewPage allows prespool approximation of the length of print jobs and may be used to alter job scheduling. The function may be called once, when the entire metafile is spooled, with the total number of pages for the print job.

This call also sets *ulPagesSpooled* in PRJINFO4 for the print job.

```
#define INCL_SPL
#define INCL_SPLBIDI
#include <os2.h>

HSPL    hspl;           /* Spool file handle returned by SplQmOpen. */
ULONG   ulPageNumber;   /* Page number of next SplQmWrite. */
ULONG   rc;             /* Return codes. */

rc = SplQmNewPage(hspl, ulPageNumber);
```

SplQmNewPage Parameter - hspl

hspl ([HSPL](#)) - input
Spool file handle returned by SplQmOpen.

SplQmNewPage Parameter - ulPageNumber

ulPageNumber ([ULONG](#)) - input
Page number of next SplQmWrite.

The first page number would be 1 (one).

SplQmNewPage Return Value - rc

rc ([ULONG](#)) - returns
Return codes.

0	Success
ERROR_INVALID_HANDLE(6)	Invalid handle given.

SplQmNewPage - Parameters

hspl ([HSPL](#)) - input
Spool file handle returned by SplQmOpen.

uiPageNumber ([ULONG](#)) - input
Page number of next SplQmWrite.

The first page number would be 1 (one).

rc ([ULONG](#)) - returns
Return codes.

0	Success
ERROR_INVALID_HANDLE(6)	Invalid handle given.

SplQmNewPage - Remarks

None.

SplQmNewPage - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Glossary](#)

SplQmOpen

SplQmOpen - Syntax

[Description:](#)

SplQmOpen opens a spool file for generating a print job. It corresponds to the [DevOpenDC](#) call.

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ          pszToken;    /* Reserved value. */
LONG         lCount;      /* Number of items. */
```

```
PQMOPENDATA    pqmdopData; /* Pointer to DevOpenStruc. */
HSPL           hspl;       /* Spooler handle. */

hspl = SplQmOpen(pszToken, lCount, pqmdopData);
```

SplQmOpen Parameter - pszToken

pszToken (**PSZ**) - input
Reserved value.

SplQmOpen Parameter - lCount

lCount (**LONG**) - input
Number of items.

This is the number of items present in the *pqmdopData* supplied. This can be shorter than the full list, if omitted items are irrelevant.

This parameter must be greater than 0.

SplQmOpen Parameter - pqmdopData

pqmdopData (**PQMOPENDATA**) - input
Pointer to DevOpenStruc.

SplQmOpen Return Value - hspl

hspl (**HSPL**) - returns
Spooler handle.

Nonzero	Spooler handle
SPL_ERROR	Error

SplQmOpen - Parameters

pszToken ([PSZ](#)) - input
Reserved value.

ICount ([LONG](#)) - input
Number of items.

This is the number of items present in the *pqmdopData* supplied. This can be shorter than the full list, if omitted items are irrelevant.

This parameter must be greater than 0.

pqmdopData ([PQMOPENDATA](#)) - input
Pointer to DevOpenStruc.

hspl ([HSPL](#)) - returns
Spooler handle.

Nonzero	Spooler handle
SPL_ERROR	Error

SplQmOpen - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_PARM (0x1303)
A parameter to the function contained invalid data.

PMERR_SPL_INV_LENGTH_OR_COUNT (0x401A)
The length or count is invalid.

PMERR_SPL_QUEUE_NOT_FOUND (0x4024)
The spooler queue definition could not be found.

PMERR_BASE_ERROR (0x2006)
An OS/2 base error has occurred. The base error code can be accessed using the OffBinaryData field of the ERRINFO structure returned by [WinGetErrorInfo](#).

SplQmOpen - Related Functions

Related Functions

- [DevOpenDC](#)
-

SplQmOpen - Example Code

This sample code will initialize a PDEVOPENSTRUC and use it to call the function.

```
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_BASE
#define INCL_ERRORS

#include <os2.h>
```

```

#include <stdio.h>
#include <stdlib.h>

VOID main()
{
    HSPL hspl;
    PDEVOPENSTRUC pdata;          /* Pointer to a DEVOPENSTRUC structure */
    PSZ pszToken = "";           /* Spooler info identifier */

    /* Allocate memory for pdata */
    if ( !DosAllocMem( &pdata, sizeof( DEVOPENSTRUC ),
        (PAG_READ|PAG_WRITE|PAG_COMMIT) ) )
    {
        /* Initialize elements of pdata */
        pdata->pszLogAddress      = "LPT1Q1";
        pdata->pszDriverName      = "IBMNULL";
        pdata->pdriv              = NULL;
        pdata->pszDataType        = "PM_Q_STD";
        pdata->pszComment         = NULL;
        pdata->pszQueueProcName   = NULL;
        pdata->pszQueueProcParams = NULL;
        pdata->pszSpoolerParams   = NULL;
        pdata->pszNetworkParams   = NULL;

        hspl = SplQmOpen( pszToken, 4L, ( PQMOPENDATA )pdata );

        if ( hspl != SPL_ERROR )      /* Good spooler handle */
        {
            printf("SplQmOpen handle is %d\n",hspl);
        }
        else
        {
            printf("SplQmOpen failed.\n");
        }
    }
}

```

SplQmOpen - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

SplQmStartDoc

SplQmStartDoc - Syntax

Description:

SplQmStartDoc starts a print job. It corresponds to the [DevEscape](#) (DEVESC_STARTDOC) call.

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

HSPL    hspl;        /* Spooler handle. */
PSZ     pszDocName;  /* Document name. */
BOOL    rc;          /* Success indicator. */

rc = SplQmStartDoc(hspl, pszDocName);
```

SplQmStartDoc Parameter - hspl

hspl ([HSPL](#)) - input
Spooler handle.

SplQmStartDoc Parameter - pszDocName

pszDocName ([PSZ](#)) - input
Document name.

This is part of the job description that is displayed to the end user by the spooler.

SplQmStartDoc Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SplQmStartDoc - Parameters

hspl ([HSPL](#)) - input
Spooler handle.

pszDocName ([PSZ](#)) - input
Document name.

This is part of the job description that is displayed to the end user by the spooler.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SplQmStartDoc - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_PARM (0x1303)
A parameter to the function contained invalid data.

PMERR_SPL_QUEUE_ERROR (0x4004)
No spooler queue supplied or found.

PMERR_ENDDOC_NOT_ISSUED (0x210B)
A request to close the spooled output without first issuing a an ENDDOC was attempted.

PMERR_SPL_INV_HSPL (0x4005)
The spooler handle is invalid.

SplQmStartDoc - Related Functions

Prerequisite Functions

- [SplQmOpen](#)

Related Functions

- [DevEscape](#)
-

SplQmStartDoc - Example Code

This function indicates the start of a print job. It allows the application to specify a document name to be associated with the print job.

Multiple print jobs can be generated, within a single queue manager open, by bracketing each job with SplQmStartDoc and [SplQmEndDoc](#).

```
#define INCL_SPL
#include <OS2.H>

HSPL hspl; /* spooler handle. */
CHAR szDocName[] = "Test Job";
CHAR szMsg[100];
HWND hwndClient;

sprintf(szMsg, "Starting job named: %s", szDocName);
WinMessageBox(HWND_DESKTOP,
             hwndClient, /* client-window handle */
```

```
szMsg,          /* body of the message */
"Printing Information", /* title of the message */
0,             /* message box id */
MB_NOICON | MB_OK); /* icon and button flags */

SplQmStartDoc(hspl,szDocName);
```

SplQmStartDoc - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

SplQmWrite

SplQmWrite - Syntax

Description:

SplQmWrite writes a buffer of data to the spool file for the print job.

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

HSPL    hspl; /* Spooler handle. */
LONG    lCount; /* Length in bytes. */
PVOID    pData; /* Buffer of data to be written to the spool file. */
BOOL    rc; /* Success indicator. */

rc = SplQmWrite(hspl, lCount, pData);
```

SplQmWrite Parameter - hspl

hspl ([HSPL](#)) - input
Spooler handle.

SplQmWrite Parameter - ICount

ICount (**LONG**) - input
Length in bytes.

This is the length of *pData*; it must not be less than 1 or greater than 65 535. Data that is longer than this must be written by two or more calls.

SplQmWrite Parameter - pData

pData (**PVOID**) - input
Buffer of data to be written to the spool file.

SplQmWrite Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SplQmWrite - Parameters

hSpl (**HSPL**) - input
Spooler handle.

ICount (**LONG**) - input
Length in bytes.

This is the length of *pData*; it must not be less than 1 or greater than 65 535. Data that is longer than this must be written by two or more calls.

pData (**PVOID**) - input
Buffer of data to be written to the spool file.

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SplQmWrite - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_PARM (0x1303)

A parameter to the function contained invalid data.

PMERR_BASE_ERROR (0x2006)

An OS/2 base error has occurred. The base error code can be accessed using the OffBinaryData field of the ERRINFO structure returned by [WinGetErrorInfo](#).

PMERR_SPL_INV_LENGTH_OR_COUNT (0x401A)

The length or count is invalid.

PMERR_SPL_QUEUE_ERROR (0x4004)

No spooler queue supplied or found.

PMERR_SPL_PRINT_ABORT (0x4008)

The job has already been aborted.

PMERR_STARTDOC_NOT_ISSUED (0x2104)

A request to write spooled output without first issuing a STARTDOC was attempted.

PMERR_SPL_CANNOT_OPEN_FILE (0x401C)

Unable to open the file.

PMERR_SPL_INV_HSPL (0x4005)

The spooler handle is invalid.

PMERR_SPL_NO_DISK_SPACE (0x4006)

There is not enough free disk space.

SplQmWrite - Related Functions

Prerequisite Functions

- [SplQmOpen](#)
 - [SplQmStartDoc](#)
-

SplQmWrite - Example Code

This function writes a buffer of data to the spool file for the print job. The size of the data buffer must not be greater than 64KB. Print jobs that exceed the maximum buffer size must be written by repeatedly calling to this function.

```
#define INCL_SPL
#include <OS2.H>

HSPL hspl; /* spooler handle. */

SplQmWrite(hspl,
           strlen("DATA"),
           (PVOID)"DATA");
```

SplQmWrite - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

SplQueryControlPanel

SplQueryControlPanel - Syntax

Description:

This function queries a specific control panel DLL for support for the printer attached to the given port.

```
#define INCL_SPL
#define INCL_SPLBIDI
#include <os2.h>

PSZ      pszComputerName; /* Name of computer connected to the printer port. */
PSZ      pszPortName;    /* Printer port name. */
PSZ      pszDeviceID;    /* Device ID, if known, for the printer connected to pszPortName. */
PSZ      pszControlPanel; /* Name of control panel to call to query support for this printer. */
ULONG    flCapabilities;  /* Capabilities requested. */
PULONG   pulOptions;      /* Option value returned by the control panel DLL. */
ULONG    rc;              /* Return codes. */

rc = SplQueryControlPanel(pszComputerName,
    pszPortName, pszDeviceID, pszControlPanel,
    flCapabilities, pulOptions);
```

SplQueryControlPanel Parameter - pszComputerName

pszComputerName ([PSZ](#)) - input

Name of computer connected to the printer port.

This parameter is NULL for printers whose port exists on the current machine.

SplQueryControlPanel Parameter - pszPortName

pszPortName ([PSZ](#)) - input
Printer port name.

The name of the printer port in *pszComputerName* used to determine if a control panel can be displayed. Following is an example of a port name:
LPT1

This will be a locally defined port name.

SplQueryControlPanel Parameter - pszDeviceID

pszDeviceID ([PSZ](#)) - input
Device ID, if known, for the printer connected to *pszPortName*.

SplQueryControlPanel Parameter - pszControlPanel

pszControlPanel ([PSZ](#)) - input
Name of control panel to call to query support for this printer.

This name can be one of the following:

- The name registered with [SplRegisterControlPanel](#)
- One of the control panel names returned by [SplGetControlPanelList](#)

This parameter can be used to determine the type of support given by a control panel in the list.

SplQueryControlPanel Parameter - flCapabilities

flCapabilities ([ULONG](#)) - input
Capabilities requested.

FL_ADMIN - 0x00000001

Set if an administrator wants the control panel.

SplQueryControlPanel Parameter - pulOptions

pulOptions ([PULONG](#)) - output

Option value returned by the control panel DLL.

Values are as follows:

OPT_CUSTOM(2)

A control panel for the specific device is available.

OPT_GENERIC(1)

A generic control panel can be displayed for the printer.

SplQueryControlPanel Return Value - rc

rc ([ULONG](#)) - returns

Return codes.

0

Success

ERROR_INVALID_NAME(123)

pszName is not the name of a control panel DLL.

ERROR_NOT_SUPPORTED(50)

A control panel for this printer is not supported by the called control panel DLL.

SplQueryControlPanel - Parameters

pszComputerName ([PSZ](#)) - input

Name of computer connected to the printer port.

This parameter is NULL for printers whose port exists on the current machine.

pszPortName ([PSZ](#)) - input

Printer port name.

The name of the printer port in *pszComputerName* used to determine if a control panel can be displayed. Following is an example of a port name:

LPT1

This will be a locally defined port name.

pszDeviceID ([PSZ](#)) - input

Device ID, if known, for the printer connected to *pszPortName*.

pszControlPanel ([PSZ](#)) - input

Name of control panel to call to query support for this printer.

This name can be one of the following:

- The name registered with [SplRegisterControlPanel](#)
- One of the control panel names returned by [SplGetControlPanelList](#)

This parameter can be used to determine the type of support given by a control panel in the list.

flCapabilities ([ULONG](#)) - input
Capabilities requested.

FL_ADMIN - 0x00000001

Set if an administrator wants the control panel.

pulOptions ([PULONG](#)) - output
Option value returned by the control panel DLL.

Values are as follows:

OPT_CUSTOM(2)

A control panel for the specific device is available.

OPT_GENERIC(1)

A generic control panel can be displayed for the printer.

rc ([ULONG](#)) - returns
Return codes.

0

Success

ERROR_INVALID_NAME(123)

pszName is not the name of a control panel DLL.

ERROR_NOT_SUPPORTED(50)

A control panel for this printer is not supported by the called control panel DLL.

SplQueryControlPanel - Remarks

None.

SplQueryControlPanel - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Glossary](#)

SplQueryDevice

SplQueryDevice - Syntax

This function retrieves information about a print device.

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where print device is to be queried. */
PSZ      pszPrintDeviceName; /* Name of Print Device. */
ULONG    ulLevel; /* Level of detail required. */
PVOID    pBuf; /* Buffer. */
ULONG    cbBuf; /* Size, in bytes, of Buffer. */
PULONG    pcbNeeded; /* Size in bytes of available information. */
SPLERR    rc; /* Return code. */

rc = SplQueryDevice(pszComputerName, pszPrintDeviceName,
    ulLevel, pBuf, cbBuf, pcbNeeded);
```

SplQueryDevice Parameter - pszComputerName

pszComputerName (PSZ) - input
Name of computer where print device is to be queried.

A NULL string specifies the local workstation.

SplQueryDevice Parameter - pszPrintDeviceName

pszPrintDeviceName (PSZ) - input
Name of Print Device.

This can specify a print device name or a port name. If *ulLevel* is 0, it must be a port name. If *ulLevel* is 2 or 3, it must be a print device name.

SplQueryDevice Parameter - ulLevel

ulLevel (ULONG) - input
Level of detail required.

This must be 0, 2 or 3.

SplQueryDevice Parameter - pBuf

pBuf (PVOID) - in/out
Buffer.

The returned contents of the buffer depend on the value specified in *ulLevel* as follows:

	<i>ulLevel</i>	Buffer Contents
0		A port name consisting of 9 characters, including a null terminator.
2		A print device name of type PSZ .
3		A PRDINFO3 structure.

SplQueryDevice Parameter - cbBuf

cbBuf ([ULONG](#)) - input
Size, in bytes, of Buffer.

It must be greater than 0.

SplQueryDevice Parameter - pcbNeeded

pcbNeeded ([PULONG](#)) - output
Size in bytes of available information.

SplQueryDevice Return Value - rc

rc ([SPLERR](#)) - returns
Return code.

- NO_ERROR (0)
No errors occurred.
- ERROR_NOT_SUPPORTED (50)
This request is not supported by the network.
- ERROR_BAD_NETPATH (53)
The network path cannot be located.
- ERROR_INVALID_PARAMETER (87)
An invalid parameter is specified.
- ERROR_INVALID_LEVEL (124)
The level parameter is invalid.
- ERROR_MORE_DATA (234)
Additional data is available.
- NERR_NetNotStarted (2102)
The network program is not started.
- NERR_BufTooSmall (2123)
The API return buffer is too small.
- NERR_DestNotFound (2152)
The print device cannot be found.
- NERR_InvalidComputer (2351)
The computer name is invalid.

SplQueryDevice - Parameters

pszComputerName (PSZ) - input

Name of computer where print device is to be queried.

A NULL string specifies the local workstation.

pszPrintDeviceName (PSZ) - input

Name of Print Device.

This can specify a print device name or a port name. If *ulLevel* is 0, it must be a port name. If *ulLevel* is 2 or 3, it must be a print device name.

ulLevel (ULONG) - input

Level of detail required.

This must be 0, 2 or 3.

pBuf (PVOID) - in/out

Buffer.

The returned contents of the buffer depend on the value specified in *ulLevel* as follows:

<i>ulLevel</i>	Buffer Contents
0	A port name consisting of 9 characters, including a null terminator.
2	A print device name of type PSZ .
3	A PRDINFO3 structure.

cbBuf (ULONG) - input

Size, in bytes, of Buffer.

It must be greater than 0.

pcbNeeded (PULONG) - output

Size in bytes of available information.

rc (SPLERR) - returns

Return code.

NO_ERROR (0)

No errors occurred.

ERROR_NOT_SUPPORTED (50)

This request is not supported by the network.

ERROR_BAD_NETPATH (53)

The network path cannot be located.

ERROR_INVALID_PARAMETER (87)

An invalid parameter is specified.

ERROR_INVALID_LEVEL (124)

The level parameter is invalid.

ERROR_MORE_DATA (234)

Additional data is available.

NERR_NetNotStarted (2102)

The network program is not started.

NERR_BufTooSmall (2123)

The API return buffer is too small.

NERR_DestNotFound (2152)

The print device cannot be found.

NERR_InvalidComputer (2351)

The computer name is invalid.

SplQueryDevice - Remarks

If *ulLevel* is 0, the port name in *pBuf* is truncated to 8 characters.

If *ulLevel* is 3, and *pBuf* cannot hold the entire [PRDINFO3](#) structure, SplQueryDevice returns NERR_BufTooSmall (2123).

To obtain the size of buffer required, call `SplQueryDevice` with the required value of *ulLevel* and a NULL buffer. The number of bytes required is returned in *pcbNeeded*.

If no job is printing on the print device, bits 2-11 of *fsStatus* in the [PRDINFO3](#) data structure are meaningless.

SplQueryDevice - Related Functions

Related Functions

- [SplCreateDevice](#)
 - [SplDeleteDevice](#)
 - [SplEnumDevice](#)
-

SplQueryDevice - Example Code

This sample code returns information for the device name that is entered at the command line. The local workstation is selected. The query is done for level 3 information.

```
#define INCL_BASE
#define INCL_DOSMEMMGR
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    SPLERR splerr ;
    ULONG  cbBuf;
    ULONG  cbNeeded ;
    ULONG  ulLevel ;
    PSZ     pszComputerName ;
    PSZ     pszPrintDeviceName ;
    PVOID   pBuf ;
    PPRDINFO3 pprd3 ;

    if (argc != 2)
    {
        printf("Syntax:  sdqry  DeviceName  \n");
        DosExit( EXIT_PROCESS , 0 ) ;
    }

    pszComputerName = (PSZ)NULL ;
    pszPrintDeviceName = argv[1];
    ulLevel = 3;
    splerr = SplQueryDevice(pszComputerName, pszPrintDeviceName,
                           ulLevel, (PVOID)NULL, 0L, &cbNeeded );
    if (splerr != NERR_BufTooSmall)
    {
        printf("SplQueryDevice Err=%ld, cbNeeded=%ld\n", splerr, cbNeeded) ;
        DosExit( EXIT_PROCESS , 0 ) ;
    }
    if (!DosAllocMem( &pBuf, cbNeeded,
                     PAG_READ|PAG_WRITE|PAG_COMMIT) ){
        cbBuf= cbNeeded ;
        splerr = SplQueryDevice(pszComputerName, pszPrintDeviceName,
                               ulLevel, pBuf, cbBuf, &cbNeeded) ;

        printf("SplQueryDevice Error=%ld, Bytes Needed=%ld\n", splerr,
               cbNeeded) ;
    }
}
```

```

        pprd3=(PPRDINFO3)pBuf;

        printf("Print Device info: name - %s\n", pprd3->pszPrinterName) ;
        printf("User Name      = %s\n", pprd3->pszUserName) ;
        printf("Logical Address= %s\n", pprd3->pszLogAddr) ;
        printf("Job ID        = %d\n", pprd3->uJobId) ;
        printf("Status        = %d\n", pprd3->fsStatus) ;
        printf("Status Comment = %s\n", pprd3->pszStatus) ;
        printf("Comment       = %s\n", pprd3->pszComment) ;
        printf("Drivers       = %s\n", pprd3->pszDrivers) ;
        printf("Time         = %d\n", pprd3->time) ;
        printf("Time Out      = %d\n", pprd3->usTimeOut) ;
        DosFreeMem(pBuf) ;
    }
    DosExit( EXIT_PROCESS , 0 ) ;
    return (splerr);
}

```

SplQueryDevice - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

SplQueryJob

SplQueryJob - Syntax

This function retrieves information about a print job.

```

#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where print job is to be queried. */
PSZ      pszQueueName;   /* Queue Name. */
ULONG    ulJob;          /* Job identification number. */
ULONG    ulLevel;        /* Level of detail required. */
PVOID    pBuf;           /* Buffer. */
ULONG    cbBuf;          /* Size, in bytes, of Buffer. */
PULONG    pcbNeeded;     /* Size in bytes of available information. */
SPLERR    rc;            /* Return code. */

rc = SplQueryJob(pszComputerName, pszQueueName,
                ulJob, ulLevel, pBuf, cbBuf, pcbNeeded);

```

SplQueryJob Parameter - pszComputerName

pszComputerName (PSZ) - input
Name of computer where print job is to be queried.

A NULL string specifies the local workstation.

SplQueryJob Parameter - pszQueueName

pszQueueName (PSZ) - input
Queue Name.

SplQueryJob Parameter - ulJob

ulJob (ULONG) - input
Job identification number.

SplQueryJob Parameter - ulLevel

ulLevel (ULONG) - input
Level of detail required.

This must be 0, 2, or 3.

SplQueryJob Parameter - pBuf

pBuf (PVOID) - in/out
Buffer.

The returned contents of the buffer depend on the value specify in *ulLevel* as follow:

<i>ulLevel</i>	Buffer Contents
0	The job identifier
2	A PRJINFO2 structure, with variable information, up to the <i>cbBuf</i> of <i>pBuf</i>
3	A PRJINFO3 structure, with variable information, up to the <i>cbBuf</i> of <i>pBuf</i> .

SplQueryJob Parameter - cbBuf

cbBuf (**ULONG**) - input
Size, in bytes, of Buffer.

It must be greater than 0.

SplQueryJob Parameter - pcbNeeded

pcbNeeded (**PULONG**) - output
Size in bytes of available information.

SplQueryJob Return Value - rc

rc (**SPLERR**) - returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
ERROR_MORE_DATA (234)	Additional data is available.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_JobNotFound (2151)	The print job does not exist.
NERR_SpoolerNotLoaded (2161)	The spooler is not running.
NERR_InvalidComputer (2351)	The computer name is invalid.

SplQueryJob - Parameters

pszComputerName (**PSZ**) - input
Name of computer where print job is to be queried.

A NULL string specifies the local workstation.

pszQueueName ([PSZ](#)) - input
Queue Name.

ulJob ([ULONG](#)) - input
Job identification number.

ulLevel ([ULONG](#)) - input
Level of detail required.

This must be 0, 2, or 3.

pBuf ([PVOID](#)) - in/out
Buffer.

The returned contents of the buffer depend on the value specify in *ulLevel* as follow:

<i>ulLevel</i>	Buffer Contents
0	The job identifier
2	A PRJINFO2 structure, with variable information, up to the <i>cbBuf</i> of <i>pBuf</i>
3	A PRJINFO3 structure, with variable information, up to the <i>cbBuf</i> of <i>pBuf</i> .

cbBuf ([ULONG](#)) - input
Size, in bytes, of Buffer.

It must be greater than 0.

pcbNeeded ([PULONG](#)) - output
Size in bytes of available information.

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
ERROR_MORE_DATA (234)	Additional data is available.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_JobNotFound (2151)	The print job does not exist.
NERR_SpoolerNotLoaded (2161)	The spooler is not running.
NERR_InvalidComputer (2351)	The computer name is invalid.

SplQueryJob - Related Functions

Related Functions

- [SplEnumJob](#)
- [SplEnumQueue](#)

- [SplQueryQueue](#)
- [SplSetJob](#)

SplQueryJob - Example Code

The following sample code will print out the information contained in a PRJINFO3 structure that is returned from a SplQueryJob call. The parameters that are entered on the command line are the computer name, queue name, and the job id.

```
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>      /* for printf call */
#include <stdlib.h>     /* for atoi call */

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    INT      splerr;
    ULONG    cbBuf ;
    ULONG    cbNeeded ;
    ULONG    ulLevel ;
    ULONG    ulJob ;
    PSZ      pszComputerName ;
    PSZ      pszQueueName ;
    PVOID     pBuf;
    PPRJINFO3 pprj3 ;

    /* Input the parameters Computer Name, Queue Name, and Job ID. Check that */
    /* three parameters have been entered along with the program name.      */
    if (argc != 4)
    {
        /* Print a message and exit if wrong number of parameters entered */
        printf("Syntax: sjqry ComputerName QueueName JobID \n");
        DosExit( EXIT_PROCESS , 0 );
    }
    /* Set the parameters to the values entered on the command line.          */
    pszComputerName = argv[1] ;
    pszQueueName = argv[2] ;
    ulJob = atoi (argv[3]);

    /* Valid levels are 0,2,and 3. Level 3 returns a PRJINFO3 structure.      */
    ulLevel = 3 ;

    /* Call the function with cbBuf equal to zero in order to get the number */
    /* of bytes needed returned in cbNeeded.                                  */
    splerr = SplQueryJob(pszComputerName,pszQueueName,ulJob,
        ulLevel, (PVOID)NULL, 0L, &cbNeeded );

    /* Only continue if the error return code is one of the two following.    */
    if (splerr == NERR_BufTooSmall || splerr == ERROR_MORE_DATA )
    {
        /* Allocate memory for the buffer(pBuf). Only continue if memory is */
        /* successfully allocated.                                           */
        if (DosAllocMem( &pBuf, cbNeeded,
            PAG_READ|PAG_WRITE|PAG_COMMIT) )
        {
            /* Set the count of bytes needed for the buffer to the value */
            /* returned in cbNeeded from the first call.                  */
            cbBuf = cbNeeded ;

            /* Make the call again with all the correct values.            */
            SplQueryJob(pszComputerName,pszQueueName,ulJob,
                ulLevel, pBuf, cbBuf, &cbNeeded) ;

            /* Set a pointer to point to the beginning of the buffer that holds */
            /* the returned structure.                                           */
            pprj3=(PPRJINFO3)pBuf;

            /* Print out the information for each element in the structure.    */

```

```

printf("Job ID      = %d\n", pprj3->uJobId);
printf("Job Priority= %d\n", pprj3->uPriority);
printf("User Name   = %s\n", pprj3->pszUserName);
printf("Position    = %d\n", pprj3->uPosition);
printf("Status      = %d\n", pprj3->fsStatus);
printf("Submitted   = %ld\n", pprj3->ulSubmitted);
printf("Size        = %ld\n", pprj3->ulSize);
printf("Comment     = %s\n", pprj3->pszComment);
printf("Document    = %s\n", pprj3->pszDocument);
printf("Notify Name  = %s\n", pprj3->pszNotifyName);
printf("Data Type    = %s\n", pprj3->pszDataType);
printf("Parms       = %s\n", pprj3->pszParms);
printf("Status      = %s\n", pprj3->pszStatus);
printf("Queue        = %s\n", pprj3->pszQueue);
printf("QProc Name   = %s\n", pprj3->pszQProcName);
printf("QProc Parms  = %s\n", pprj3->pszQProcParms);
printf("Driver Name  = %s\n", pprj3->pszDriverName);
printf("Printer Name= %s\n", pprj3->pszPrinterName);

/* If pDriverData is NULL, then we can not access any data.      */
if (pprj3->pDriverData)
{
    printf("  pDriverData->cb          - %ld\n",
           (ULONG)pprj3->pDriverData->cb);
    printf("  pDriverData->lVersion    - %ld\n",
           (ULONG)pprj3->pDriverData->lVersion) ;
    printf("  pDriverData->szDeviceName - %s\n",
           pprj3->pDriverData->szDeviceName) ;
}
printf("/n");

/* Free memory that we allocated.                                */
DosFreeMem(pBuf) ;
}
}
else
{
    /* If we are here than we have an error code. Print it out.    */
    printf("SplQueryJob Error=%ld,Bytes Needed=%ld\n",splerr, cbNeeded);
}
DosExit( EXIT_PROCESS , 0 ) ;
return(splerr);
}

```

SplQueryJob - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

SplQueryPath

SplQueryPath - Syntax

Description:

This function returns the path to get printer drivers or port drivers.

When a new printer is being installed that requires a printer driver not yet installed, the print object calls this function to get the path to the print drivers.

```
#define INCL_SPL
#define INCL_SPLBIDI
#include <os2.h>

PSZ      pszComputerName;    /* Name of the computer for which a network print object is being created. */
PSZ      pszRemoteQueueName; /* Name of the remote queue the network print object is referencing. */
ULONG    ulLevel;           /* Level of information to return. */
PVOID    pBuf;              /* Receives the path to printer drivers for level 0. */
ULONG    cbBuf;              /* Length of buffer pointed to by pBuf, in bytes. */
PULONG    pcbNeeded;         /* Receives length, in bytes, of buffer needed to return path. */
ULONG    rc;                 /* Return codes. */

rc = SplQueryPath(pszComputerName, pszRemoteQueueName,
                  ulLevel, pBuf, cbBuf, pcbNeeded);
```

SplQueryPath Parameter - pszComputerName

pszComputerName (**PSZ**) - input

Name of the computer for which a network print object is being created.

This value can be NULL.

SplQueryPath Parameter - pszRemoteQueueName

pszRemoteQueueName (**PSZ**) - input

Name of the remote queue the network print object is referencing.

This value can be NULL.

SplQueryPath Parameter - ulLevel

ulLevel (**ULONG**) - input

Level of information to return.

This must be 0 (zero).

SplQueryPath Parameter - pBuf

pBuf (**PVOID**) - output
Receives the path to printer drivers for level 0.

SplQueryPath Parameter - cbBuf

cbBuf (**ULONG**) - input
Length of buffer pointed to by *pBuf*, in bytes.

SplQueryPath Parameter - pcbNeeded

pcbNeeded (**PULONG**) - output
Receives length, in bytes, of buffer needed to return path.

SplQueryPath Return Value - rc

rc (**ULONG**) - returns
Return codes.

0	Success
ERROR_NOT_SUPPORTED(50)	A path to printer drivers cannot be found.
NERR_BufTooSmall(2123)	Must call again with a bigger buffer.

SplQueryPath - Parameters

pszComputerName (**PSZ**) - input
Name of the computer for which a network print object is being created.

This value can be NULL.

pszRemoteQueueName (**PSZ**) - input
Name of the remote queue the network print object is referencing.

This value can be NULL.

uiLevel (**ULONG**) - input
Level of information to return.

This must be 0 (zero).

pBuf ([PVOID](#)) - output
Receives the path to printer drivers for level 0.

cbBuf ([ULONG](#)) - input
Length of buffer pointed to by *pBuf*, in bytes.

pcbNeeded ([PULONG](#)) - output
Receives length, in bytes, of buffer needed to return path.

rc ([ULONG](#)) - returns
Return codes.

0	Success
ERROR_NOT_SUPPORTED(50)	A path to printer drivers cannot be found.
NERR_BufTooSmall(2123)	Must call again with a bigger buffer.

SplQueryPath - Remarks

The spooler will look in the following places for the path to printer drivers:

1. The spooler will first look in the system .INI file for applicationName PM_SPOOLER_DRVSHARE. If found, the spooler will return the keyValue for keyName *pszComputerName* if it exists, or for the first keyName that exists under the applicationName.
2. The spooler will check the *pszComputerName* field for a share called PRINTDRV.
3. The spooler will check the user's logon domain LS:*ALIAS for a share called PRINTDRV.

If a path is found, the spooler will return the path to the caller. The path can be one of the following:

- An extended UNC path (LS:\\SERVER1\\PRINTDRV)
- A UNC path with additional subdirectories defined (LS:\\SERVER1\\PRINTDRV\\DIR1\\NEXT)
- A fully qualified pathname (Z:\\OS2\\SHARED)

The print object will use the path as follows:

- Look in the root of the path for packed files (.DR_)
- Look in the root of the path for unpacked files (.DRV)
- Look in the subdirectory PMDD_n from path, where *n* is the OS2 printer driver diskette number.

SplQueryPath - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

SplQueryPort

SplQueryPort - Syntax

Description:

This function queries information about a printer port.

```
#define INCL_SPL
#define INCL_SPLBIDI
#include <os2.h>

PSZ      pszComputerName; /* Name of the computer connected to the port. */
PSZ      pszPortName;    /* Name of the port to query. */
ULONG    ulLevel;        /* Level of information to return. */
PVOID    pBuf;           /* Receives the PRPORTINFO2 data structure. */
ULONG    cbBuf;          /* Length of buffer pointed to by pBuf, in bytes. */
PULONG    pcbNeeded;     /* Receives length, in bytes, of buffer needed to return all the port data. */
ULONG    rc;             /* Return codes. */

rc = SplQueryPort(pszComputerName, pszPortName,
                  ulLevel, pBuf, cbBuf, pcbNeeded);
```

SplQueryPort Parameter - pszComputerName

pszComputerName ([PSZ](#)) - input

Name of the computer connected to the port.

This value can be NULL for a local machine.

SplQueryPort Parameter - pszPortName

pszPortName ([PSZ](#)) - input

Name of the port to query.

SplQueryPort Parameter - ulLevel

ulLevel (**ULONG**) - input
Level of information to return.

This must be 2.

SplQueryPort Parameter - pBuf

pBuf (**PVOID**) - output
Receives the PRPORTINFO2 data structure.

SplQueryPort Parameter - cbBuf

cbBuf (**ULONG**) - input
Length of buffer pointed to by *pBuf*, in bytes.

SplQueryPort Parameter - pcbNeeded

pcbNeeded (**PULONG**) - output
Receives length, in bytes, of buffer needed to return all the port data.

SplQueryPort Return Value - rc

rc (**ULONG**) - returns
Return codes.

0	Success
ERROR_INVALID_LEVEL(124)	<i>ulLevel</i> is not 2.
ERROR_INVALID_PARAMETER(87)	An invalid parameter was specified; most likely an invalid <i>pBuf</i> value.
ERROR_MORE_DATA(234)	Not all data could fit in <i>pBuf</i> .
ERROR_NOT_SUPPORTED(50)	Command not supported on <i>pszComputerName</i> .
NERR_BufTooSmall(2123)	No data could fit in <i>pBuf</i> .

NERR_DestNotFound(2152)

The port name does not exist.

SplQueryPort - Parameters

pszComputerName ([PSZ](#)) - input

Name of the computer connected to the port.

This value can be NULL for a local machine.

pszPortName ([PSZ](#)) - input

Name of the port to query.

ulLevel ([ULONG](#)) - input

Level of information to return.

This must be 2.

pBuf ([PVOID](#)) - output

Receives the PRPORTINFO2 data structure.

cbBuf ([ULONG](#)) - input

Length of buffer pointed to by *pBuf*, in bytes.

pcbNeeded ([PULONG](#)) - output

Receives length, in bytes, of buffer needed to return all the port data.

rc ([ULONG](#)) - returns

Return codes.

0 Success

ERROR_INVALID_LEVEL(124)
ulLevel is not 2.

ERROR_INVALID_PARAMETER(87)
An invalid parameter was specified; most likely an invalid *pBuf* value.

ERROR_MORE_DATA(234)
Not all data could fit in *pBuf*.

ERROR_NOT_SUPPORTED(50)
Command not supported on *pszComputerName*.

NERR_BufTooSmall(2123)
No data could fit in *pBuf*.

NERR_DestNotFound(2152)
The port name does not exist.

SplQueryPort - Remarks

None.

SplQueryPort - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

SplQueryQueue

SplQueryQueue - Syntax

This function supplies information about a print queue, and, optionally, about the jobs in it.

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where queue is to be queried. */
PSZ      pszQueueName;   /* Queue name. */
ULONG    ulLevel;        /* Level of detail required. */
PVOID    pBuf;           /* Buffer. */
ULONG    cbBuf;          /* Size, in bytes, of Buffer. */
PULONG   pcbNeeded;      /* Size in bytes of available information. */
SPLERR   rc;             /* Return code. */

rc = SplQueryQueue(pszComputerName, pszQueueName,
                  ulLevel, pBuf, cbBuf, pcbNeeded);
```

SplQueryQueue Parameter - pszComputerName

pszComputerName ([PSZ](#)) - input
Name of computer where queue is to be queried.

A NULL string specifies the local workstation.

SplQueryQueue Parameter - pszQueueName

pszQueueName ([PSZ](#)) - input
Queue name.

SplQueryQueue Parameter - ulLevel

ulLevel ([ULONG](#)) - input
Level of detail required.

This must be 3, 4, 5, 6 or 7.

SplQueryQueue Parameter - pBuf

pBuf ([PVOID](#)) - in/out
Buffer.

The returned contents of the buffer depend on the value specified in *ulLevel* as follows:

<i>ulLevel</i>	Buffer Contents
3	A PRQINFO3 structure, with associated variable information up to <i>cbBuf</i> . The <i>fsType</i> bit PRQ3_TYPE_APPDEFAULT is set for the application default queue only.
4	A PRQINFO3 structure, with associated variable information, and an array of PRJINFO2 structures, one for each job in the queue, up to <i>cbBuf</i> .
5	A queue name of type PSZ .
6	A PRQINFO6 structure, with associated variable information up to <i>cbBuf</i> .
7	A PRQINFO3 structure, with associated variable information, and an array of PRJINFO4 structures, one for each job in the queue, up to <i>cbBuf</i> .

SplQueryQueue Parameter - cbBuf

cbBuf ([ULONG](#)) - input
Size, in bytes, of Buffer.

It must be greater than 0.

SplQueryQueue Parameter - pcbNeeded

pcbNeeded ([PULONG](#)) - output
Size in bytes of available information.

SplQueryQueue Return Value - rc

rc (SPLERR) - returns
Return code.

NO_ERROR (0)
No errors occurred.

ERROR_ACCESS_DENIED (5)
Access is denied.

ERROR_NOT_SUPPORTED (50)
This request is not supported by the network.

ERROR_BAD_NETPATH (53)
The network path cannot be located.

ERROR_INVALID_PARAMETER (87)
An invalid parameter is specified.

ERROR_INVALID_LEVEL (124)
The level parameter is invalid.

ERROR_MORE_DATA (234)
Additional data is available.

NERR_NetNotStarted (2102)
The network program is not started.

NERR_BufTooSmall (2123)
The API return buffer is too small.

NERR_QNotFound (2150)
The printer queue does not exist.

NERR_InvalidComputer (2351)
The computer name is invalid.

SplQueryQueue - Parameters

pszComputerName (PSZ) - input
Name of computer where queue is to be queried.

A NULL string specifies the local workstation.

pszQueueName (PSZ) - input
Queue name.

ulLevel (ULONG) - input
Level of detail required.

This must be 3, 4, 5, 6 or 7.

pBuf (PVOID) - in/out
Buffer.

The returned contents of the buffer depend on the value specified in *ulLevel* as follows:

<i>ulLevel</i>	Buffer Contents
3	A PRQINFO3 structure, with associated variable information up to <i>cbBuf</i> . The <i>fsType</i> bit PRQ3_TYPE_APPDEFAULT is set for the application default queue only.
4	A PRQINFO3 structure, with associated variable information, and an array of PRJINFO2 structures, one for each job in the queue, up to <i>cbBuf</i> .
5	A queue name of type PSZ .
6	A PRQINFO6 structure, with associated variable information up to <i>cbBuf</i> .
7	A PRQINFO3 structure, with associated variable information, and an array of PRJINFO4 structures, one for each job in the queue, up to <i>cbBuf</i> .

cbBuf (ULONG) - input

Size, in bytes, of Buffer.

It must be greater than 0.

pcbNeeded ([PULONG](#)) - output
Size in bytes of available information.

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)
No errors occurred.
ERROR_ACCESS_DENIED (5)
Access is denied.
ERROR_NOT_SUPPORTED (50)
This request is not supported by the network.
ERROR_BAD_NETPATH (53)
The network path cannot be located.
ERROR_INVALID_PARAMETER (87)
An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)
The level parameter is invalid.
ERROR_MORE_DATA (234)
Additional data is available.
NERR_NetNotStarted (2102)
The network program is not started.
NERR_BufTooSmall (2123)
The API return buffer is too small.
NERR_QNotFound (2150)
The printer queue does not exist.
NERR_InvalidComputer (2351)
The computer name is invalid.

SplQueryQueue - Remarks

If *ulLevel* is 3 or 4, and *pBuf* cannot hold the entire [PRQINFO3](#) structure, SplQueryQueue returns NERR_BufTooSmall (2123).

If *ulLevel* is 6, and *pBuf* cannot hold the entire [PRQINFO6](#) structure, SplQueryQueue returns NERR_BufTooSmall (2123).

If *ulLevel* is 4, and *pBuf* cannot hold all the available [PRJINFO2](#) structures, SplQueryQueue returns ERROR_MORE_DATA (234).

To obtain the size of buffer required, call SplQueryQueue with the required value of *ulLevel*, *cbBuf* set to 0 (zero), and a NULL buffer. The number of bytes required is returned in *pcbNeeded*.

SplQueryQueue - Related Functions

Related Functions

- [SplEnumQueue](#)
 - [SplQueryJob](#)
 - [SplSetJob](#)
 - [SplSetQueue](#)
-

SplQueryQueue - Example Code

This sample code queries the local workstation for a queue name that is entered at the command prompt. The query is done at level 4 which returns returns in the buffer information in a PRQINFO3 structure and follows this with PRJINFO2 structures - one for each job in the queue.

```
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>

INT main (argc, argv)
INT argc;
CHAR *argv[];

{
    ULONG splerr;
    ULONG cbBuf;
    ULONG cbNeeded;
    ULONG ulLevel;
    ULONG i;
    USHORT uJobCount;
    PSZ pszComputerName;
    PSZ pszQueueName;
    PVOID pBuf;
    PPRJINFO2 prj2;
    PPRQINFO3 prq3 ;

    if (argc != 2)
    {
        printf("Syntax: setqryq QueueName \n");
        DosExit(EXIT_PROCESS, 0);
    }

    pszComputerName = (PSZ)NULL;
    pszQueueName = argv[1];
    ulLevel = 4L;
    splerr = SplQueryQueue(pszComputerName,
                           pszQueueName,
                           ulLevel,
                           (PVOID)NULL,
                           0L,
                           &cbNeeded);

    if (splerr != NERR_BufTooSmall ||
        splerr != ERROR_MORE_DATA)
    {
        printf("SplQueryQueue Error=%ld, cbNeeded=%ld\n", splerr, cbNeeded);
        DosExit(EXIT_PROCESS, 0);
    }

    if (!DosAllocMem(&pBuf, cbNeeded,
                    PAG_READ |
                    PAG_WRITE |
                    PAG_COMMIT))
    {
        cbBuf = cbNeeded;
        splerr = SplQueryQueue(pszComputerName,
                               pszQueueName,
                               ulLevel,
                               pBuf,
                               cbBuf,
                               &cbNeeded);

        prq3=(PPRQINFO3)pBuf;
        printf("Queue info: name- %s\n", prq3->pszName);
        if (prq3->fsType & PRQ3_TYPE_APPDEFAULT)
            printf(" This is the application default print queue\n");
        printf(" priority - %d starttime - %d endtime - %d fsType - %X\n",
               prq3->uPriority,
               prq3->uStartTime,
               prq3->uUntilTime,
               prq3->fsType);

        printf(" pszSepFile - %s\n", prq3->pszSepFile);
        printf(" pszPrProc - %s\n", prq3->pszPrProc);
        printf(" pszParms - %s\n", prq3->pszParms);
        printf(" pszComment - %s\n", prq3->pszComment);
        printf(" pszPrinters - %s\n", prq3->pszPrinters);
        printf(" pszDriverName - %s\n", prq3->pszDriverName);
    }
}
```

```

if (prq3->pDriverData)
{
    printf("    pDriverData->cb          - %ld\n",
        (ULONG)prq3->pDriverData->cb);
    printf("    pDriverData->lVersion      - %ld\n",
        (ULONG)prq3->pDriverData->lVersion);
    printf("    pDriverData->szDeviceName - %s\n",
        prq3->pDriverData->szDeviceName);
}
/* Store the job count for use later in the for loop */
uJobCount = prq3->cJobs;
printf("Job count in this queue is %d\n\n",uJobCount);

/* Increment the pointer to the PRQINFO3 structure      */
/* so that it points to the first structure after itself */
prq3++;

/* Cast the prq3 pointer to point to a PRJINFO2 structure, */
/* and set a pointer to point to that place                */
prj2=(PPRJINFO2)prq3;
for (i=0 ; i<uJobCount ;i++)
{
    printf("Job ID      = %d\n",    prj2->uJobId);
    printf("Priority    = %d\n",    prj2->uPriority);
    printf("User Name   = %s\n",    prj2->pszUserName);
    printf("Position    = %d\n",    prj2->uPosition);
    printf("Status      = %d\n",    prj2->fsStatus);
    printf("Submitted   = %ld\n",    prj2->ulSubmitted);
    printf("Size        = %ld\n",    prj2->ulSize);
    printf("Comment     = %s\n",    prj2->pszComment);
    printf("Document    = %s\n\n",prj2->pszDocument);

    /* Increment the pointer to point to the next structure */
    prj2++;
} /* endfor */
DosFreeMem(pBuf);
}
DosExit(EXIT_PROCESS, 0);
return (splerr);
}

```

SplQueryQueue - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

SplRegister

SplRegister - Syntax

Description:

This function allows applications to register for notification of events about printers. This function may be used for job accounting purposes or by printer drivers to keep aware of the operation of the print subsystem.

```
#define INCL_SPL
#define INCL_SPLBIDI
#include <os2.h>

PSZ      pszComputerName; /* Printer server name to register for alert notification. */
PSZ      pszName;         /* Case-sensitive name of printer port to register for alerts. */
HWND     hwndNotify;      /* Window handle to get notify messages. */
ULONG     ulCategory;     /* Category of alert to register. */
ULONG     ulType;         /* Type of alert to enable for the given category. */
PULONG    pulHandle;      /* Pointer to spooler alert handle. */
ULONG     rc;             /* Return codes. */

rc = SplRegister(pszComputerName, pszName,
                hwndNotify, ulCategory, ulType, pulHandle);
```

SplRegister Parameter - pszComputerName

pszComputerName ([PSZ](#)) - input

Printer server name to register for alert notification.

The field is used by an application running on a client machine to register for alerts on a print server. To register locally, this field must be NULL.

SplRegister Parameter - pszName

pszName ([PSZ](#)) - input

Case-sensitive name of printer port to register for alerts.

This value can be a virtual port name.

SplRegister Parameter - hwndNotify

hwndNotify ([HWND](#)) - input

Window handle to get notify messages.

The message posted is as follows:

Message ID is WM_SPOOLER_ALERT(0x00d8)

MP1 will define the type of alert as follows:

```

/* Alert type - SHORT1FROMMP(MP1) bits 0-7 */
#define SPLMSG_SPECIAL_ALERT      0x00FC
#define SPLMSG_EXTENDED_ALERT    0x00FD
#define SPLMSG_CORE_ALERT_MASK   0x00FF

/* Extended alert type - SHORT1FROMMP(MP1) bits 8-15 */
#define SPLMSG_EXTENDED_TYPE_MASK 0xFF00

```

If the object type is SPLMSG_EXTENDED_ALERT, then this is an extended alert. The extended alert category is SHORT2FROMMP(MP1). The type of alert in the given extended alert category is SHORT1FROMMP(MP1) bits 8-15. This limits the notification given to applications registered for extended alerts to only the specific categories that are 0xFFFF or less and to only category types that are 0xFF or less.

If the object type is SPLMSG_SPECIAL_ALERT, then this is a special alert generated by the spooler and SHORT2FROMMP(MP1) will contain one of the following messages:

SPLMSG_CONNECTION_LOST - 0x0002
Connection to computer terminated; all alerts disabled for given computer.

SPLMSG_PORT_TERMINATED - 0x0003
Registration terminated for port.

SPLMSG_SPOOLER_DISABLED - 0x0001
Spooler disabled; all alerts for given computer are disabled.

If the object type is less than SPLMSG_SPECIAL_ALERT(0xFC), then this is a core alert. The specific core alert posted will be: SHORT1FROMMP(MP1) bits 0-7(SPLMSG_CORE_ALERT_MASK)

If the core alert is PRTALERT_TYPE_PAGE_PRINTED(9), then SHORT1FROMMP(MP1) bits 8-15(SPLMSG_PAGE_PRINTED_MASK) will be the page number printed. If the page number is above 255, the value hex 'FF' will be put in this byte field.

If the core alert is **not** PRTALERT_TYPE_PAGE_PRINTED(9), then SHORT1FROMMP(MP1) bits 8-15(SPLMSG_CORE_SEVERITY_MASK) will be the severity of the alert. SHORT2FROMMP(MP1) will be the spooler job ID if this is a core alert for a print job.

The core alert type will be one of the following:

Value	Description
1	PRTALERT_TYPE_INPUT Input alert (Example: out of paper).
2	PRTALERT_TYPE_OUTPUT Output alert (Example: output bin full).
3	PRTALERT_TYPE_JAM Paper jam alert.
4	PRTALERT_TYPE_OPERATOR Operator intervention required (Example: print ribbon jam).
5	PRTALERT_TYPE_CONFIG_CHANGE Configuration changed (Example: cartridge removed).
6	PRTALERT_TYPE_SUPPLIES Supplies alert (Example: out of toner).
7	PRTALERT_TYPE_JOB_START Job started printing.
8	PRTALERT_TYPE_JOB_STACKED Job completed and stacked.
9	PRTALERT_TYPE_PAGE_PRINTED Page printed.
10	PRTALERT_TYPE_JOB_CANCELLED Job cancelled inside printer.
11	PRTALERT_TYPE_JOB_HELD Job held inside printer.

12	PRTALERT_TYPE_COVER_OPEN Cover or panel open.
13	PRTALERT_TYPE_POWER_ON Printer powered on.
14	PRTALERT_TYPE_RESET Printer reset.
15	PRTALERT_TYPE_ONLINE Printer now online.
16	PRTALERT_TYPE_OFFLINE Printer now offline.
17	PRTALERT_TYPE_TIMED_ALERT Timed alert.
18	PRTALERT_TYPE_COMMUNICATION_PROBLEM Communication problem.
19	PRTALERT_TYPE_COMM_STATUS_CHANGED Communication status changed. BIDI_Q_PORT should be issued to get current state.
0xFF	PRTALERT_TYPE_OTHER Other alert type (0xFFFFFFFF).

The severity of the problem will be one of the following:

Value	Description
00	PRTALERT_SEV_INFORMATIONAL
01	PRTALERT_SEV_WARNING
02	PRTALERT_SEV_ERROR Printing stopped due to error.
03	PRTALERT_SEV_SERVICE Service is required.
04 - FD	Reserved
FE	PRTALERT_SEV_OTHER Other (0xFE)
FF	PRTALERT_SEV_UNKNOWN Unknown (0xFF)

MP2 contents are as follows:

SHORT1FROMMP(MP1) is the atomized print server name.

If the notification is from the local spooler, this value will be zero.

SHORT2FROMMP(MP2) is the atomized print queue name when the message involves a print job (alert values 7 - 11); otherwise, it will be the atomized printer port name, if applicable.

SplRegister Parameter - ulCategory

ulCategory (**ULONG**) - input

Category of alert to register.

This value can be the core alert category or an extended alert category as defined below:

PRTALERT_CATEGORY_CORE - 0x00000001
Core alert category

PRTALERT_CATEGORY_EXTENDED - 0x0000D000
Beginning of the extended alert categories. Protocol converters should limit the category range to between 0x0000D000 and 0x0000DFFF.

PRTALERT_CATEGORY_EXT_MAX - 0x0000DFFF
End of the extended alert categories.

If the protocol converter does not support the given category, ERROR_INVALID_CATEGORY(117) is returned.

SplRegister Parameter - ulType

ulType (ULONG) - input
Type of alert to enable for the given category.

If the category is PRTALERT_CATEGORY_CORE, this is a bit-field that defines the additional core alerts to enable. The core alerts corresponding to the bits set in the *ulType* field will be enabled, in addition to any currently enabled core alerts. If the protocol converter does not understand one of the bits that are set, ERROR_INVALID_FLAG_NUMBER(186) should be returned.

If the printer is not capable of setting one of the core alerts whose bit is set, and if any core alerts can be enabled, a Success return code should be returned.

If the category is **not** PRTALERT_CATEGORY_CORE, then this is a value that defines the specific extended alert to enable.

Bit definitions for core alerts are as follows:

Bit	Description
0	PRTALERT_CORE_INPUT - 0x00000001 Input alert (Example: out of paper).
1	PRTALERT_CORE_OUTPUT - 0x00000002 Output alert (Example: output bin is full).
2	PRTALERT_CORE_JAM - 0x00000004 Paper jam alert.
3	PRTALERT_CORE_OPERATOR - 0x00000008 Operator intervention required (Example: print ribbon jam).
4	PRTALERT_CORE_CONFIG_CHANGE - 0x00000010 Configuration changed (Example: cartridge removed).
5	PRTALERT_CORE_SUPPLIES - 0x00000020 Supplies alert (Example: out of toner).
6	PRTALERT_CORE_JOB_START - 0x00000040 Job started printing.
7	PRTALERT_CORE_JOB_STACKED - 0x00000080 Job completed and stacked.
8	PRTALERT_CORE_PAGE_PRINTED - 0x00000100 Page printed.
9	PRTALERT_CORE_JOB_CANCELLED - 0x00000200 Job cancelled inside printer.
10	PRTALERT_CORE_JOB_HELD - 0x00000400 Job held inside printer.
11	PRTALERT_CORE_COVER_OPEN - 0x00000800

	Cover or panel open.
12	PRTALERT_CORE_POWER_ON - 0x00001000 Printer powered on.
13	PRTALERT_CORE_RESET - 0x00002000 Printer reset.
14	PRTALERT_CORE_ONLINE - 0x00004000 Printer now online.
15	PRTALERT_CORE_OFFLINE - 0x00008000 Printer now offline.
16	PRTALERT_CORE_TIMED_ALERT - 0x00010000 Timed alert.
17	PRTALERT_CORE_COMMUNICATION_PROBLEM - 0x00020000 Communication problem.
18	PRTALERT_CORE_COMM_STATUS_CHANGED - 0x00040000 Communication status changed.

SplRegister Parameter - pulHandle

pulHandle ([PULONG](#)) - in/out
Pointer to spooler alert handle.

For initial registration, the value pointed to by this parameter should be 0 (zero). If an application wants to make better use of spooler resources, the application can use the same returned spooler handle for all of its alert registration.

SplRegister Return Value - rc

rc ([ULONG](#)) - returns
Return codes.

0	Success
ERROR_INVALID_CATEGORY(117)	The protocol converter does not support the extended alert category.
ERROR_INVALID_FLAG_NUMBER(186)	The protocol converter did not understand a core alert that was set.
ERROR_INVALID_HANDLE(6)	The value pointed to by <i>pulHandle</i> is not a valid spooler alert handle. The value at <i>*pulHandle</i> should be set to 0 (zero) to allocate a new spooler alert handle.
ERROR_INVALID_PARAMETER(87)	An invalid buffer given.
ERROR_NOT_SUPPORTED(50)	The spooler on the given print server cannot support alert notification.
PMERR_SPL_SPOOLER_NOT_INSTALLED(0x4009)	The spooler is not enabled.

SplRegister - Parameters

pszComputerName (PSZ) - input

Printer server name to register for alert notification.

The field is used by an application running on a client machine to register for alerts on a print server. To register locally, this field must be NULL.

pszName (PSZ) - input

Case-sensitive name of printer port to register for alerts.

This value can be a virtual port name.

hwndNotify (HWND) - input

Window handle to get notify messages.

The message posted is as follows:

Message ID is WM_SPOOLER_ALERT(0x00d8)

MP1 will define the type of alert as follows:

```
/* Alert type - SHORT1FROMMP(MP1) bits 0-7 */
#define SPLMSG_SPECIAL_ALERT      0x00FC
#define SPLMSG_EXTENDED_ALERT    0x00FD
#define SPLMSG_CORE_ALERT_MASK   0x00FF

/* Extended alert type - SHORT1FROMMP(MP1) bits 8-15 */
#define SPLMSG_EXTENDED_TYPE_MASK 0xFF00
```

If the object type is SPLMSG_EXTENDED_ALERT, then this is an extended alert. The extended alert category is SHORT2FROMMP(MP1). The type of alert in the given extended alert category is SHORT1FROMMP(MP1) bits 8-15. This limits the notification given to applications registered for extended alerts to only the specific categories that are 0xFFFF or less and to only category types that are 0xFF or less.

If the object type is SPLMSG_SPECIAL_ALERT, then this is a special alert generated by the spooler and SHORT2FROMMP(MP1) will contain one of the following messages:

SPLMSG_CONNECTION_LOST - 0x0002
Connection to computer terminated; all alerts disabled for given computer.

SPLMSG_PORT_TERMINATED - 0x0003
Registration terminated for port.

SPLMSG_SPOOLER_DISABLED - 0x0001
Spooler disabled; all alerts for given computer are disabled.

If the object type is less than SPLMSG_SPECIAL_ALERT(0xFC), then this is a core alert. The specific core alert posted will be: SHORT1FROMMP(MP1) bits 0-7(SPLMSG_CORE_ALERT_MASK)

If the core alert is PRTALERT_TYPE_PAGE_PRINTED(9), then SHORT1FROMMP(MP1) bits 8-15(SPLMSG_PAGE_PRINTED_MASK) will be the page number printed. If the page number is above 255, the value hex 'FF' will be put in this byte field.

If the core alert is **not** PRTALERT_TYPE_PAGE_PRINTED(9), then SHORT1FROMMP(MP1) bits 8-15(SPLMSG_CORE_SEVERITY_MASK) will be the severity of the alert. SHORT2FROMMP(MP1) will be the spooler job ID if this is a core alert for a print job.

The core alert type will be one of the following:

Value	Description
1	PRTALERT_TYPE_INPUT Input alert (Example: out of paper).

2	PRTALERT_TYPE_OUTPUT Output alert (Example: output bin full).
3	PRTALERT_TYPE_JAM Paper jam alert.
4	PRTALERT_TYPE_OPERATOR Operator intervention required (Example: print ribbon jam).
5	PRTALERT_TYPE_CONFIG_CHANGE Configuration changed (Example: cartridge removed).
6	PRTALERT_TYPE_SUPPLIES Supplies alert (Example: out of toner).
7	PRTALERT_TYPE_JOB_START Job started printing.
8	PRTALERT_TYPE_JOB_STACKED Job completed and stacked.
9	PRTALERT_TYPE_PAGE_PRINTED Page printed.
10	PRTALERT_TYPE_JOB_CANCELLED Job cancelled inside printer.
11	PRTALERT_TYPE_JOB_HELD Job held inside printer.
12	PRTALERT_TYPE_COVER_OPEN Cover or panel open.
13	PRTALERT_TYPE_POWER_ON Printer powered on.
14	PRTALERT_TYPE_RESET Printer reset.
15	PRTALERT_TYPE_ONLINE Printer now online.
16	PRTALERT_TYPE_OFFLINE Printer now offline.
17	PRTALERT_TYPE_TIMED_ALERT Timed alert.
18	PRTALERT_TYPE_COMMUNICATION_PROBLEM Communication problem.
19	PRTALERT_TYPE_COMM_STATUS_CHANGED Communication status changed. BIDI_Q_PORT should be issued to get current state.
0xFF	PRTALERT_TYPE_OTHER Other alert type (0xFFFFFFFF).

The severity of the problem will be one of the following:

Value	Description
00	PRTALERT_SEV_INFORMATIONAL
01	PRTALERT_SEV_WARNING
02	PRTALERT_SEV_ERROR Printing stopped due to error.
03	PRTALERT_SEV_SERVICE Service is required.
04 - FD	Reserved

FE	PRTALERT_SEV_OTHER Other (0xFE)
FF	PRTALERT_SEV_UNKNOWN Unknown (0xFF)

MP2 contents are as follows:

SHORT1FROMMP(MP1) is the atomized print server name.

If the notification is from the local spooler, this value will be zero.

SHORT2FROMMP(MP2) is the atomized print queue name when the message involves a print job (alert values 7 - 11); otherwise, it will be the atomized printer port name, if applicable.

ulCategory (ULONG) - input

Category of alert to register.

This value can be the core alert category or an extended alert category as defined below:

PRTALERT_CATEGORY_CORE - 0x00000001
Core alert category

PRTALERT_CATEGORY_EXTENDED - 0x0000D000
Beginning of the extended alert categories. Protocol converters should limit the category range to between 0x0000D000 and 0x0000DFFF.

PRTALERT_CATEGORY_EXT_MAX - 0x0000DFFF
End of the extended alert categories.

If the protocol converter does not support the given category, ERROR_INVALID_CATEGORY(117) is returned.

ulType (ULONG) - input

Type of alert to enable for the given category.

If the category is PRTALERT_CATEGORY_CORE, this is a bit-field that defines the additional core alerts to enable. The core alerts corresponding to the bits set in the *ulType* field will be enabled, in addition to any currently enabled core alerts. If the protocol converter does not understand one of the bits that are set, ERROR_INVALID_FLAG_NUMBER(186) should be returned.

If the printer is not capable of setting one of the core alerts whose bit is set, and if any core alerts can be enabled, a Success return code should be returned.

If the category is **not** PRTALERT_CATEGORY_CORE, then this is a value that defines the specific extended alert to enable.

Bit definitions for core alerts are as follows:

Bit	Description
0	PRTALERT_CORE_INPUT - 0x00000001 Input alert (Example: out of paper).
1	PRTALERT_CORE_OUTPUT - 0x00000002 Output alert (Example: output bin is full).
2	PRTALERT_CORE_JAM - 0x00000004 Paper jam alert.
3	PRTALERT_CORE_OPERATOR - 0x00000008 Operator intervention required (Example: print ribbon jam).
4	PRTALERT_CORE_CONFIG_CHANGE - 0x00000010 Configuration changed (Example: cartridge removed).
5	PRTALERT_CORE_SUPPLIES - 0x00000020 Supplies alert (Example: out of toner).
6	PRTALERT_CORE_JOB_START - 0x00000040 Job started printing.
7	PRTALERT_CORE_JOB_STACKED - 0x00000080 Job completed and stacked.
8	PRTALERT_CORE_PAGE_PRINTED - 0x00000100

	Page printed.
9	PRTALERT_CORE_JOB_CANCELLED - 0x00000200 Job cancelled inside printer.
10	PRTALERT_CORE_JOB_HELD - 0x00000400 Job held inside printer.
11	PRTALERT_CORE_COVER_OPEN - 0x00000800 Cover or panel open.
12	PRTALERT_CORE_POWER_ON - 0x00001000 Printer powered on.
13	PRTALERT_CORE_RESET - 0x00002000 Printer reset.
14	PRTALERT_CORE_ONLINE - 0x00004000 Printer now online.
15	PRTALERT_CORE_OFFLINE - 0x00008000 Printer now offline.
16	PRTALERT_CORE_TIMED_ALERT - 0x00010000 Timed alert.
17	PRTALERT_CORE_COMMUNICATION_PROBLEM - 0x00020000 Communication problem.
18	PRTALERT_CORE_COMM_STATUS_CHANGED - 0x00040000 Communication status changed.

pulHandle (**PULONG**) - in/out
Pointer to spooler alert handle.

For initial registration, the value pointed to by this parameter should be 0 (zero). If an application wants to make better use of spooler resources, the application can use the same returned spooler handle for all of its alert registration.

rc (**ULONG**) - returns
Return codes.

0	Success
ERROR_INVALID_CATEGORY(117)	The protocol converter does not support the extended alert category.
ERROR_INVALID_FLAG_NUMBER(186)	The protocol converter did not understand a core alert that was set.
ERROR_INVALID_HANDLE(6)	The value pointed to by <i>pulHandle</i> is not a valid spooler alert handle. The value at <i>*pulHandle</i> should be set to 0 (zero) to allocate a new spooler alert handle.
ERROR_INVALID_PARAMETER(87)	An invalid buffer given.
ERROR_NOT_SUPPORTED(50)	The spooler on the given print server cannot support alert notification.
PMERR_SPL_SPOOLER_NOT_INSTALLED(0x4009)	The spooler is not enabled.

SpIRegister - Remarks

Note:

Registration is **not** maintained across reboots.

Any new registration is in addition to existing registration(s) for the spooler handle passed in (if any handle was passed).

To remove registration, use [SplUnRegister](#).

For each spooler alert handle, only the **last** *hwnd/Notify* passed in with the alert handle will be posted with alert messages.

That is, if you register window handle X and receive alert handle 1, then register for more alerts using window handle Y and alert handle 1, window handle Y will get alerts for alert handle 1, but window handle X will not get any alert messages.

Registration for alerts from a client on a print server requires that the user logged onto the client have authority to open a named pipe on the server. This access is normally granted to those users who have access to the print queue on a server.

SplRegister - Topics

Select an item:

Syntax

Parameters

Returns

Remarks

Glossary

SplRegisterControlPanel

SplRegisterControlPanel - Syntax

Description:

This function registers a DLL with the spooler that can display a control panel for printers. By default, all protocol converters will be checked for control panel support, so converters must not register themselves using this API.

[illegible]

SplRegisterControlPanel Parameter - pszName

pszName ([PSZ](#)) - input
Name of control panel.

This name must not conflict with any installed protocol converter name or control panel name.

SplRegisterControlPanel Parameter - pszDllPath

pszDllPath ([PSZ](#)) - input
Fully qualified path to the DLL that exports the control panel APIs.

SplRegisterControlPanel Parameter - ulFlags

ulFlags ([ULONG](#)) - input
Must be 0 (zero).

SplRegisterControlPanel Parameter - pszReserved

pszReserved ([PSZ](#)) - input
Must be NULL.

SplRegisterControlPanel Parameter - ulVersion

ulVersion ([ULONG](#)) - input
Must be 0 (zero).

SplRegisterControlPanel Return Value - rc

rc ([ULONG](#)) - returns
Return codes.

0	Success
ERROR_INVALID_LEVEL(124)	<i>ulVersion</i> not 0.
ERROR_INVALID_NAME(123)	<i>pszName</i> is already used by a protocol converter or another control panel DLL.
ERROR_INVALID_PARAMETER(87)	<i>ulFlags</i> not 0 or invalid parameter given.
ERROR_MOD_NOT_FOUND(126)	Invalid path to DLL.
ERROR_PROC_NOT_FOUND(127)	<i>pszDllPath</i> does not export the two APIs for control panels (apiref refid=spqucp form=textonly. and SplDisplayControlPanel).

SplRegisterControlPanel - Parameters

pszName ([PSZ](#)) - input
Name of control panel.

This name must not conflict with any installed protocol converter name or control panel name.

pszDllPath ([PSZ](#)) - input
Fully qualified path to the DLL that exports the control panel APIs.

ulFlags ([ULONG](#)) - input
Must be 0 (zero).

pszReserved ([PSZ](#)) - input
Must be NULL.

ulVersion ([ULONG](#)) - input
Must be 0 (zero).

rc ([ULONG](#)) - returns
Return codes.

0	Success
ERROR_INVALID_LEVEL(124)	<i>ulVersion</i> not 0.
ERROR_INVALID_NAME(123)	<i>pszName</i> is already used by a protocol converter or another control panel DLL.
ERROR_INVALID_PARAMETER(87)	<i>ulFlags</i> not 0 or invalid parameter given.
ERROR_MOD_NOT_FOUND(126)	Invalid path to DLL.
ERROR_PROC_NOT_FOUND(127)	<i>pszDllPath</i> does not export the two APIs for control panels (apiref refid=spqucp form=textonly. and SplDisplayControlPanel).

SplRegisterControlPanel - Remarks

A control panel need register itself only once; registration is maintained across reboots.

SplRegisterControlPanel - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

SplReleaseJob

SplReleaseJob - Syntax

This function releases a held print job.

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where job is to be continued. */
PSZ      pszQueueName;    /* Queue Name. */
ULONG    ulJob;           /* Job identification number. */
SPLERR   rc;              /* Return code. */

rc = SplReleaseJob(pszComputerName, pszQueueName,
                  ulJob);
```

SplReleaseJob Parameter - pszComputerName

pszComputerName ([PSZ](#)) - input
Name of computer where job is to be continued.

A NULL string specifies the local workstation.

SplReleaseJob Parameter - pszQueueName

pszQueueName (PSZ) - input
Queue Name.

SplReleaseJob Parameter - ulJob

ulJob (ULONG) - input
Job identification number.

SplReleaseJob Return Value - rc

rc (SPLERR) - returns
Return code.

- NO_ERROR (0)
No errors occurred.
- ERROR_ACCESS_DENIED (5)
Access is denied.
- ERROR_NOT_SUPPORTED (50)
This request is not supported by the network.
- ERROR_BAD_NETPATH (53)
The network path cannot be located.
- NERR_NetNotStarted (2102)
The network program is not started.
- NERR_JobNotFound (2151)
The print job does not exist.
- NERR_SpoolerNotLoaded (2161)
The spooler is not running.
- NERR_JobInvalidState (2164)
This operation cannot be performed on the print job in its current state.
- NERR_InvalidComputer (2351)
The computer name is invalid.

SplReleaseJob - Parameters

pszComputerName (PSZ) - input
Name of computer where job is to be continued.

A NULL string specifies the local workstation.

pszQueueName (PSZ) - input
Queue Name.

ulJob (ULONG) - input
Job identification number.

rc (SPLERR) - returns
Return code.

- NO_ERROR (0)
No errors occurred.
- ERROR_ACCESS_DENIED (5)

Access is denied.
ERROR_NOT_SUPPORTED (50)
This request is not supported by the network.
ERROR_BAD_NETPATH (53)
The network path cannot be located.
NERR_NetNotStarted (2102)
The network program is not started.
NERR_JobNotFound (2151)
The print job does not exist.
NERR_SpoolerNotLoaded (2161)
The spooler is not running.
NERR_JobInvalidState (2164)
This operation cannot be performed on the print job in its current state.
NERR_InvalidComputer (2351)
The computer name is invalid.

SplReleaseJob - Remarks

Any job can be released by a user with administrator privilege.

A job created locally can be released locally regardless of user privilege level, but it can be released remotely only by a user with administrator privilege.

A remote job can be released by a user without administrator privilege only if the user identification of the person initiating the request is the same as the user identification of the person who created the job.

SplReleaseJob - Related Functions

Related Functions

- [SplEnumJob](#)
 - [SplHoldJob](#)
 - [SplQueryJob](#)
-

SplReleaseJob - Example Code

This sample code will release the job id that is entered at the prompt.

```
#define INCL_BASE
#define INCL_SPL
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>          /* for printf function */
#include <stdlib.h>         /* for atoi function  */

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    SPLERR splerr ;
    ULONG  ulJob ;
    PSZ     pszComputerName = NULL ;
    PSZ     pszQueueName = NULL ;

    /* Get job id from the input argument */
    ulJob = atoi(argv[1]);
```

```

/* Call the function to do the release. If an error is returned, print it. */
splerr=SplReleaseJob( pszComputerName, pszQueueName, ulJob);
switch (splerr)
{
case NO_ERROR:
    printf("Job %d was released.\n",ulJob);
    break;
case NERR_JobNotFound:
    printf("Job does not exist.\n");
    break;
case NERR_JobInvalidState:
    printf("This operation can't be performed on the print Job.\n");
    break;
default:
    printf("Errorcode = %ld\n",splerr);
} /* endswitch */
DosExit( EXIT_PROCESS , 0 ) ;
argc;
return (splerr);
}

```

SplReleaseJob - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

SplReleaseQueue

SplReleaseQueue - Syntax

This function releases a held print queue.

```

#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where queue is to be continued. */
PSZ      pszQueueName;    /* Queue name. */
SPLERR   rc;              /* Return code. */

rc = SplReleaseQueue(pszComputerName, pszQueueName);

```

SplReleaseQueue Parameter - pszComputerName

pszComputerName ([PSZ](#)) - input
Name of computer where queue is to be continued.

A NULL string specifies the local workstation.

SplReleaseQueue Parameter - pszQueueName

pszQueueName ([PSZ](#)) - input
Queue name.

SplReleaseQueue Return Value - rc

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_QNotFound (2150)	The printer queue does not exist.
NERR_InvalidComputer (2351)	The computer name is invalid.

SplReleaseQueue - Parameters

pszComputerName ([PSZ](#)) - input
Name of computer where queue is to be continued.

A NULL string specifies the local workstation.

pszQueueName ([PSZ](#)) - input
Queue name.

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	

This request is not supported by the network.
ERROR_BAD_NETPATH (53)
 The network path cannot be located.
NERR_NetNotStarted (2102)
 The network program is not started.
NERR_QNotFound (2150)
 The printer queue does not exist.
NERR_InvalidComputer (2351)
 The computer name is invalid.

SplReleaseQueue - Remarks

This function releases a queue that has been held by a [SplHoldQueue](#) function, or disabled by an error on the queue. It does not affect an active print queue.

To release a queue on a remote server requires administrator privilege on the remote server.

SplReleaseQueue - Related Functions

Related Functions

- [SplEnumQueue](#)
 - [SplHoldQueue](#)
 - [SplQueryQueue](#)
-

SplReleaseQueue - Example Code

This sample code will release the local queue that is entered at the prompt.

```
#define INCL_SPL
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>      /* for printf function */

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    SPLERR splerr ;
    PSZ     pszComputerName = NULL ;
    PSZ     pszQueueName ;

    /* Get queue name from the input argument. */
    pszQueueName = argv[1];

    /* Call the function to do the release. If an error is returned, print it. */
    splerr=SplReleaseQueue(pszComputerName, pszQueueName);
    if (splerr != 0L)
    {
        switch (splerr)
        {
            case NERR_QNotFound:
                printf("Queue does not exist.\n");
                break;
            case NERR_SpoolerNotLoaded:
                printf("The Spooler is not running.\n");
                break;
        }
    }
}
```

```

        default:
            printf("Errorcode = %ld\n",splerr);
        } /* endswitch */
    }
    else
    {
        printf("Queue %s was released.\n",pszQueueName);
    } /* endif */
    DosExit( EXIT_PROCESS , 0 ) ;
    return (splerr);
}

```

SplReleaseQueue - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

SplSetDevice

SplSetDevice - Syntax

This function modifies the configuration of a print device.

```

#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where print device is to be modified. */
PSZ      pszPrintDeviceName; /* Name of Print Device. */
ULONG    ulLevel; /* Level of detail required. */
PVOID    pBuf; /* Buffer. */
ULONG    cbBuf; /* Size, in bytes, of Buffer. */
ULONG    ulParmNum; /* Parameter number. */
SPLERR    rc; /* Return code. */

rc = SplSetDevice(pszComputerName, pszPrintDeviceName,
    ulLevel, pBuf, cbBuf, ulParmNum);

```

SplSetDevice Parameter - pszComputerName

pszComputerName (PSZ) - input
Name of computer where print device is to be modified.

A NULL string specifies the local workstation.

SplSetDevice Parameter - pszPrintDeviceName

pszPrintDeviceName (PSZ) - input
Name of Print Device.

SplSetDevice Parameter - ulLevel

ulLevel (ULONG) - input
Level of detail required.

This must be 3.

SplSetDevice Parameter - pBuf

pBuf (PVOID) - input
Buffer.

If *ulParmNum* is 0, this parameter must contain a complete PRJINFO3 structure Otherwise,it must contain a valid new value for the parameter of the PRJINFO3 structure indicated in *ulParmNum*

SplSetDevice Parameter - cbBuf

cbBuf (ULONG) - input
Size, in bytes, of Buffer.

It must be grater than 0.

SplSetDevice Parameter - ulParmNum

ulParmNum (ULONG) - input
Parameter number.

Specifies either that the entire [PRDINFO3](#) structure is to be modified, or only one specific parameter is to be modified.

If *ulParmNum* is 0, *pBuf* must contain a complete [PRDINFO3](#) structure. Otherwise, *pBuf* must contain a new valid value for the parameter to be modified.

The following are the possible values for this parameter:

Parameter	Constant (Value)
<i>pszLogAddr</i>	PRD_LOGADDR_PARMNUM (3)
<i>pszComment</i>	PRD_COMMENT_PARMNUM (7)
<i>pszDrivers</i>	PRD_DRIVERS_PARMNUM (8)
<i>usTimeOut</i>	PRD_TIMEOUT_PARMNUM (10)

SplSetDevice Return Value - rc

rc ([SPLERR](#)) - returns

Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_DestNotFound (2152)	The print device cannot be found.
NERR_DestInvalidState (2162)	This operation cannot be performed on the print device.
NERR_SpoolNoMemory (2165)	A spooler memory allocation failure occurred.
NERR_DriverNotFound (2166)	The device driver does not exist.
NERR_BadDev (2341)	The requested device is invalid.
NERR_InvalidComputer (2351)	The computer name is invalid.

SplSetDevice - Parameters

pszComputerName ([PSZ](#)) - input

Name of computer where print device is to be modified.

A NULL string specifies the local workstation.

pszPrintDeviceName ([PSZ](#)) - input

Name of Print Device.

ulLevel (**ULONG**) - input
Level of detail required.

This must be 3.

pBuf (**PVOID**) - input
Buffer.

If *ulParmNum* is 0, this parameter must contain a complete **PRJINFO3** structure. Otherwise, it must contain a valid new value for the parameter of the **PRJINFO3** structure indicated in *ulParmNum*.

cbBuf (**ULONG**) - input
Size, in bytes, of Buffer.

It must be greater than 0.

ulParmNum (**ULONG**) - input
Parameter number.

Specifies either that the entire **PRDINFO3** structure is to be modified, or only one specific parameter is to be modified.

If *ulParmNum* is 0, *pBuf* must contain a complete **PRDINFO3** structure. Otherwise, *pBuf* must contain a new valid value for the parameter to be modified.

The following are the possible values for this parameter:

Parameter	Constant (Value)
<i>pszLogAddr</i>	PRD_LOGADDR_PARMNUM (3)
<i>pszComment</i>	PRD_COMMENT_PARMNUM (7)
<i>pszDrivers</i>	PRD_DRIVERS_PARMNUM (8)
<i>usTimeOut</i>	PRD_TIMEOUT_PARMNUM (10)

rc (**SPLERR**) - returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_DestNotFound (2152)	The print device cannot be found.
NERR_DestInvalidState (2162)	This operation cannot be performed on the print device.
NERR_SpoolNoMemory (2165)	A spooler memory allocation failure occurred.
NERR_DriverNotFound (2166)	The device driver does not exist.
NERR_BadDev (2341)	The requested device is invalid.
NERR_InvalidComputer (2351)	The computer name is invalid.

SplSetDevice - Remarks

This function allows modification of a print device and its connection to a logical address. To disconnect a print device from a port, use

ulLevel=3, *ulParmNum*=3, and *pBuf* is a NULL string.

To modify a print device on a remote server requires administrator privilege.

SplSetDevice - Related Functions

Related Functions

- [SplEnumDevice](#)
 - [SplEnumDriver](#)
 - [SplEnumPort](#)
 - [SplEnumPrinter](#)
 - [SplQueryDevice](#)
-

SplSetDevice - Example Code

This sample code first gets a device name from the command line. It then prompts the user for a parameter number and a value associated with it.

```
#define INCL_BASE
#define INCL_DOSMEMMGR
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>           /* for printf function */
#include <string.h>          /* for strlen function */
#include <stdlib.h>          /* for atoi function */

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    CHAR    bufValue[2]={0};
    CHAR    bufInput[128]={0};
    ULONG    splerr ;
    ULONG    cbBuf ;
    ULONG    ulParmNum ;
    USHORT    usParm;
    PSZ    pszComputerName ;
    PSZ    pszPrintDeviceName ;
    PVOID    pBuf;

    if (argc != 2)
    {
        printf("Syntax:  sdset DeviceName \n");
        DosExit( EXIT_PROCESS , 0 ) ;
    }
    pszComputerName = (PSZ)NULL ;

    /* Set the print device name to the value from the command line. */
    pszPrintDeviceName = argv[1];

    /* Get the parameter and the value. Store them in buffers. */
    printf("Enter Parameter number to be modified\n");
    gets(&bufValue[0]);
    printf("Enter new parameter value \n");
    gets(&bufInput[0]);

    /* Convert the input parmnum to a ULONG. */
    ulParmNum = atoi(&bufValue[0]);

    switch (ulParmNum)
    {
        case 10:
```

```

        /* Determine the size of the buffer. */
        cbBuf = sizeof(PUSHORT);

        /* Convert the input parameter to a USHORT. */
        usParm =(USHORT)atoi(&bufInput[0]);

        /* Point the buffer to the value. */
        pBuf = &usParm;
        break;
    case 3:
    case 7:
    case 8:
        /* Determine the size of the buffer. */
        cbBuf = strlen(&bufInput[0])+1;

        /* Point the buffer to the value. */
        pBuf = (PSZ)&bufInput;
        break;
    default:
        printf("Invalid number\n");
        DosExit( EXIT_PROCESS , 0 ) ;
        break;
}

/* Make the call. */
splerr = SplSetDevice(pszComputerName,pszPrintDeviceName,3L,
                    pBuf,cbBuf,ulParmNum) ;

/* Print out the result. */
printf("SplSetDevice Err= %ld, Parameter= %d, cbBuf= %ld ,ulParmNum= %ld\n",
        splerr, usParm, cbBuf, ulParmNum);

DosExit( EXIT_PROCESS , 0 ) ;
return (splerr);
}

```

SplSetDevice - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

SplSetJob

SplSetJob - Syntax

This function modifies the instructions for a print job.

```
#define INCL_SPL /* Or use INCL_PM, */
```

```
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where job is to be modified. */
PSZ      pszQueueName;   /* Queue Name. */
ULONG    ulJob;           /* Job identification number. */
ULONG    ulLevel;         /* Level of detail required. */
PVOID    pBuf;            /* Buffer. */
ULONG    cbBuf;           /* Size, in bytes, of Buffer. */
ULONG    ulParmNum;       /* Parameter number. */
SPLERR    rc;             /* Return code. */

rc = SplSetJob(pszComputerName, pszQueueName,
               ulJob, ulLevel, pBuf, cbBuf, ulParmNum);
```

SplSetJob Parameter - pszComputerName

pszComputerName (PSZ) - input
Name of computer where job is to be modified.

A NULL string specifies the local workstation.

SplSetJob Parameter - pszQueueName

pszQueueName (PSZ) - input
Queue Name.

SplSetJob Parameter - ulJob

ulJob (ULONG) - input
Job identification number.

SplSetJob Parameter - ulLevel

ulLevel (ULONG) - input
Level of detail required.

This must be 3.

SplSetJob Parameter - pBuf

pBuf ([PVOID](#)) - input
Buffer.

If *ulParmNum* is 0, this parameter must contain a complete [PRJINFO3](#) structure. Otherwise, it must contain a valid new value for the parameter of the [PRJINFO3](#) structure, indicated in *ulParmNum*

SplSetJob Parameter - cbBuf

cbBuf ([ULONG](#)) - input
Size, in bytes, of Buffer.

It must be greater than 0.

SplSetJob Parameter - ulParmNum

ulParmNum ([ULONG](#)) - input
Parameter number.

Specifies either that the entire [PRJINFO3](#) structure is to be modified, or that only one specific parameter is to be modified.

If *ulParmNum* is 0, *pBuf* must contain a complete [PRJINFO3](#) structure. Otherwise, *pBuf* must contain a new valid value for the parameter to be modified. The following are the possible values for this parameter:

Parameter	Value
<i>pszNotifyName</i>	PRJ_NOTIFYNAME_PARMNUM (3)
<i>pszDataType</i>	PRJ_DATATYPE_PARMNUM (4)
<i>pszParms</i>	PRJ_PARS_PARMNUM (5)
<i>uPosition</i>	PRJ_POSITION_PARMNUM (6)
<i>pszComment</i>	PRJ_COMMENT_PARMNUM (11)
<i>pszDocument</i>	PRJ_DOCUMENT_PARMNUM (12)
<i>pszStatus</i>	PRJ_STATUSCOMMENT_PARMNUM (13)
<i>uPriority</i>	PRJ_PRIORITY_PARMNUM (14)
<i>pszQProcParms</i>	PRJ_PROCPARMS_PARMNUM (16)
<i>pDriverData</i>	PRJ_DRIVERDATA_PARMNUM (18)

uPosition must be given the appropriate value, as follows:

Value	Change
0	No change
1	Move to first place
>1	Move to this position, or if the specified value is greater than the number of jobs in the queue, move to the end of the queue.

SplSetJob Return Value - rc

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)
No errors occurred.

ERROR_ACCESS_DENIED (5)
Access is denied.

ERROR_NOT_SUPPORTED (50)
This request is not supported by the network.

ERROR_BAD_NETPATH (53)
The network path cannot be located.

ERROR_INVALID_PARAMETER (87)
An invalid parameter is specified.

ERROR_INVALID_LEVEL (124)
The level parameter is invalid.

NERR_NetNotStarted (2102)
The network program is not started.

NERR_BufTooSmall (2123)
The API return buffer is too small.

NERR_JobNotFound (2151)
The print job does not exist.

NERR_SpoolerNotLoaded (2161)
The spooler is not running.

NERR_JobInvalidState (2164)
This operation cannot be performed on the print job in its current state.

NERR_SpoolNoMemory (2165)
A spooler memory allocation failure occurred.

NERR_DriverNotFound (2166)
The device driver does not exist.

NERR_ProcNotFound (2168)
The queue processor is not installed.

NERR_InvalidComputer (2351)
The computer name is invalid.

SplSetJob - Parameters

pszComputerName (PSZ) - input
Name of computer where job is to be modified.

A NULL string specifies the local workstation.

pszQueueName (PSZ) - input
Queue Name.

ulJob (ULONG) - input
Job identification number.

ulLevel (ULONG) - input
Level of detail required.

This must be 3.

pBuf (PVOID) - input
Buffer.

If *ulParmNum* is 0, this parameter must contain a complete [PRJINFO3](#) structure. Otherwise, it must contain a valid new value for the parameter of the [PRJINFO3](#) structure. indicated in *ulParmNum*

cbBuf (ULONG) - input
Size, in bytes, of Buffer.

It must be greater than 0.

ulParmNum (ULONG) - input
Parameter number.

Specifies either that the entire [PRJINFO3](#) structure is to be modified, or that only one specific parameter is to be modified.

If *ulParmNum* is 0, *pBuf* must contain a complete [PRJINFO3](#) structure. Otherwise, *pBuf* must contain a new valid value for the parameter to be modified. The following are the possible values for this parameter:

Parameter	Value
<i>pszNotifyName</i>	PRJ_NOTIFYNAME_PARMNUM (3)
<i>pszDataType</i>	PRJ_DATATYPE_PARMNUM (4)
<i>pszParms</i>	PRJ_PARS_PARMNUM (5)
<i>uPosition</i>	PRJ_POSITION_PARMNUM (6)
<i>pszComment</i>	PRJ_COMMENT_PARMNUM (11)
<i>pszDocument</i>	PRJ_DOCUMENT_PARMNUM (12)
<i>pszStatus</i>	PRJ_STATUSCOMMENT_PARMNUM (13)
<i>uPriority</i>	PRJ_PRIORITY_PARMNUM (14)
<i>pszQProcParms</i>	PRJ_PROCPARMS_PARMNUM (16)
<i>pDriverData</i>	PRJ_DRIVERDATA_PARMNUM (18)

uPosition must be given the appropriate value, as follows:

Value	Change
0	No change
1	Move to first place
>1	Move to this position, or if the specified value is greater than the number of jobs in the queue, move to the end of the queue.

rc ([SPLERR](#)) - returns

Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_JobNotFound (2151)	The print job does not exist.
NERR_SpoolerNotLoaded (2161)	The spooler is not running.
NERR_JobInvalidState (2164)	This operation cannot be performed on the print job in its current state.
NERR_SpoolNoMemory (2165)	A spooler memory allocation failure occurred.
NERR_DriverNotFound (2166)	The device driver does not exist.
NERR_ProcNotFound (2168)	The queue processor is not installed.
NERR_InvalidComputer (2351)	The computer name is invalid.

SplSetJob - Remarks

The job priority is changed, if necessary, to the priority of the next job after the new position. If the spooler is restarted, the order in which jobs are put on the queue depends on the priority and age of the job. This order may be different from the order following the SplSetJob call.

Users without administrator privilege can use SplSetJob only for jobs created when the same user name was logged on. They can move jobs backwards only, and cannot increase the job priority above the queue priority.

A job created locally has no associated user name, and any user can set information locally for the job. Only an administrator can set

information for a job on a remote server.

SplSetJob - Related Functions

Related Functions

- [SplEnumJob](#)
 - [SplQueryJob](#)
 - [SplQueryQueue](#)
-

SplSetJob - Example Code

This sample code first gets a queue name and a jobid from the command prompt. It then prompts the user to enter a parameter number and a value for that number.

```
#define INCL_BASE
#define INCL_DOSMEMMGR
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    CHAR    bufValue[2]={0};
    CHAR    bufInput[128]={0};
    ULONG   splerr ;
    ULONG   cbBuf;
    ULONG   ulParmNum ;
    ULONG   ulJob ;
    USHORT  usParm;
    PSZ     pszComputerName ;
    PSZ     pszQueueName ;
    PVOID   pBuf;

    if (argc != 3)
    {
        printf("Syntax:  sjset  QueueName  JobID  \n");
        DosExit( EXIT_PROCESS , 0 ) ;
    }
    pszComputerName = (PSZ)NULL ;

    /* Set values to those entered at the prompt. */
    pszQueueName = argv[1] ;
    ulJob = atoi (argv[2]);

    /* Request a parameter and the associated value. Store them in buffers. */
    printf("Enter Parameter number to be modified\n");
    gets(&bufValue[0]);
    printf("Enter new parameter value \n");
    gets(&bufInput[0]);

    /* Convert the ParmNum to a ULONG. */
    ulParmNum = atoi(&bufValue[0]);
    switch (ulParmNum)
    {
        case 6:
        case 14:
            /* Calculate size of buffer needed if this is the parameter.*/
            cbBuf = sizeof(PUSHORT);
```



```

        /* Convert input parameter into a USHORT. */
        usParm =(USHORT)atoi(&bufInput[0]);

        /* Point pBuf to the value. */
        pBuf = &usParm;
        break;
    case 3:
    case 4:
    case 5:
    case 11:
    case 12:
    case 16:
        /* Calculate size of buffer needed if this is the parameter.*/
        cbBuf = strlen(&bufInput[0])+1;

        /* Point pBuf to the value. */
        pBuf = (PSZ)&bufInput;
        break;
    case 18:
        printf("In order to keep code simple, this is not implemented.");
        break;
    default:
        printf("Invalid number\n");
}

splerr = SplSetJob(pszComputerName,pszQueueName,ulJob,3L,
                  pBuf,cbBuf,ulParmNum) ;
if ( !splerr)
    printf("Parameter was set.");
else
    printf("SplSetJob Error= %ld, Parameter= %d, Buf= %ld ,ParmNum= %ld\n",
          splerr, usParm, cbBuf, ulParmNum);
DosExit( EXIT_PROCESS , 0 ) ;
return (splerr) ;
}

```

SplSetJob - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

SplSetPort

SplSetPort - Syntax

Description:

This function sets the BIDI capabilities of a printer port.

This function is used by the print object or other applications to enable or disable BIDI communications with a printer. If the port driver is capable of BIDI communications, SplSetPort can override the port driver and disable BIDI with the printer. This is useful in case there are problems with the BIDI components.

```
#define INCL_SPL
#define INCL_SPLBIDI
#include <os2.h>

PSZ      pszComputerName; /* Name of the computer connected to the port. */
PSZ      pszPortName;    /* Name of the port to set. */
ULONG    ulLevel;        /* Level of information to set. */
PVOID    pBuf;           /* Buffer containing new port setting. */
ULONG    cbBuf;          /* Length of buffer pointed to by pBuf, in bytes. */
ULONG    ulParmNum;      /* Parameter number used to change port settings. */
ULONG    rc;             /* Return codes. */

rc = SplSetPort(pszComputerName, pszPortName,
               ulLevel, pBuf, cbBuf, ulParmNum);
```

SplSetPort Parameter - pszComputerName

pszComputerName (**PSZ**) - input
Name of the computer connected to the port.

This value can be NULL for a local machine.

SplSetPort Parameter - pszPortName

pszPortName (**PSZ**) - input
Name of the port to set.

SplSetPort Parameter - ulLevel

ulLevel (**ULONG**) - input
Level of information to set.

This must be 2.

SplSetPort Parameter - pBuf

pBuf (**PVOID**) - input

Buffer containing new port setting.

SplSetPort Parameter - cbBuf

cbBuf (**ULONG**) - input
Length of buffer pointed to by *pBuf*, in bytes.

SplSetPort Parameter - ulParmNum

ulParmNum (**ULONG**) - input
Parameter number used to change port settings.

Parameter number must be one of the following:

Value	Description								
1	PRPO_PORT_DRIVER Port driver. This will be the new port driver for the printer port.								
2	PRPO_PROTOCOL_CNV Protocol converter. This will override the values returned by BIDI_Q_PORT. The spooler will use this protocol converter when using this port in BIDI mode.								
3	PRPO_MODE BIDI mode of the port. The ULONG value pointed to by <i>pBuf</i> must be one of the following values: <table><tr><th>Value</th><th>Description</th></tr><tr><td>1</td><td>PRPORT_AUTODETECT Auto-detect mode. The spooler will use the results of BIDI_Q_PORT to decide if BIDI should be enabled and which protocol converter to use. This is the default.</td></tr><tr><td>2</td><td>PRPORT_DISABLE_BIDI BIDI disabled. The spooler will not use the printer in BIDI mode. This overrides the port driver.</td></tr><tr><td>3</td><td>PRPORT_ENABLE_BIDI BIDI enabled. This is set only if an application knows a printer is capable of using a protocol converter, but the port driver does not know which converter to use. The application must also supply the protocol converter name when setting a port to this mode.</td></tr></table>	Value	Description	1	PRPORT_AUTODETECT Auto-detect mode. The spooler will use the results of BIDI_Q_PORT to decide if BIDI should be enabled and which protocol converter to use. This is the default.	2	PRPORT_DISABLE_BIDI BIDI disabled. The spooler will not use the printer in BIDI mode. This overrides the port driver.	3	PRPORT_ENABLE_BIDI BIDI enabled. This is set only if an application knows a printer is capable of using a protocol converter, but the port driver does not know which converter to use. The application must also supply the protocol converter name when setting a port to this mode.
Value	Description								
1	PRPORT_AUTODETECT Auto-detect mode. The spooler will use the results of BIDI_Q_PORT to decide if BIDI should be enabled and which protocol converter to use. This is the default.								
2	PRPORT_DISABLE_BIDI BIDI disabled. The spooler will not use the printer in BIDI mode. This overrides the port driver.								
3	PRPORT_ENABLE_BIDI BIDI enabled. This is set only if an application knows a printer is capable of using a protocol converter, but the port driver does not know which converter to use. The application must also supply the protocol converter name when setting a port to this mode.								
4	PRPO_PRIORITY Priority. Currently not supported.								

SplSetPort Return Value - rc

rc (**ULONG**) - returns
Return codes.

0	Success
ERROR_INVALID_PARAMETER(87)	An invalid parameter was specified; most likely an invalid <i>pBuf</i> value or an invalid <i>ulParmNum</i> value.
ERROR_NOT_SUPPORTED(50)	Command not supported on <i>pszComputerName</i> .
NERR_DestNotFound(2152)	The port name does not exist.

SplSetPort - Parameters

pszComputerName (PSZ) - input

Name of the computer connected to the port.

This value can be NULL for a local machine.

pszPortName (PSZ) - input

Name of the port to set.

ulLevel (ULONG) - input

Level of information to set.

This must be 2.

pBuf (PVOID) - input

Buffer containing new port setting.

cbBuf (ULONG) - input

Length of buffer pointed to by *pBuf*, in bytes.

ulParmNum (ULONG) - input

Parameter number used to change port settings.

Parameter number must be one of the following:

Value	Description
1	PRPO_PORT_DRIVER Port driver. This will be the new port driver for the printer port.
2	PRPO_PROTOCOL_CNV Protocol converter. This will override the values returned by BIDI_Q_PORT. The spooler will use this protocol converter when using this port in BIDI mode.
3	PRPO_MODE BIDI mode of the port. The ULONG value pointed to by <i>pBuf</i> must be one of the following values:
Value	Description
1	PRPORT_AUTODETECT Auto-detect mode. The spooler will use the results of BIDI_Q_PORT to decide if BIDI should be enabled and which protocol converter to use. This is the default.
2	PRPORT_DISABLE_BIDI BIDI disabled. The spooler will not use the printer in BIDI mode. This overrides the port driver.
3	PRPORT_ENABLE_BIDI BIDI enabled. This is set only if an application knows a printer is

capable of using a protocol converter, but the port driver does not know which converter to use. The application must also supply the protocol converter name when setting a port to this mode.

4	PRPO_PRIORITY Priority. Currently not supported.
rc (ULONG) - returns Return codes.	
0	Success
ERROR_INVALID_PARAMETER(87)	An invalid parameter was specified; most likely an invalid <i>pBuf</i> value or an invalid <i>ulParmNum</i> value.
ERROR_NOT_SUPPORTED(50)	Command not supported on <i>pszComputerName</i> .
NERR_DestNotFound(2152)	The port name does not exist.

SplSetPort - Remarks

None.

SplSetPort - Topics

Select an item:

- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Glossary](#)
-

SplSetQueue

SplSetQueue - Syntax

This function modifies the configuration of a print queue.

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>

PSZ      pszComputerName; /* Name of computer where queue is to be modified. */
```

```
PSZ      pszQueueName;      /* Queue name. */
ULONG    ulLevel;           /* Level of detail required. */
PVOID    pBuf;              /* Buffer. */
ULONG    cbBuf;             /* Size, in bytes, of Buffer. */
ULONG    ulParmNum;         /* Parameter number. */
SPLERR    rc;               /* Return code. */
```

```
rc = SplSetQueue(pszComputerName, pszQueueName,
                 ulLevel, pBuf, cbBuf, ulParmNum);
```

SplSetQueue Parameter - pszComputerName

pszComputerName ([PSZ](#)) - input

Name of computer where queue is to be modified.

A NULL string specifies the local workstation.

SplSetQueue Parameter - pszQueueName

pszQueueName ([PSZ](#)) - input

Queue name.

SplSetQueue Parameter - ulLevel

ulLevel ([ULONG](#)) - input

Level of detail required.

This must be 3 or 6.

SplSetQueue Parameter - pBuf

pBuf ([PVOID](#)) - input

Buffer.

If *ulParmNum* is 0, this parameter must contain a complete [PRQINFO3](#) or [PRQINFO6](#) structure. Otherwise, it must contain a valid new value for the parameter of the [PRQINFO3](#) or the [PRQINFO6](#) structure indicated in *ulParmNum*.

SplSetQueue Parameter - cbBuf

cbBuf (**ULONG**) - input
Size, in bytes, of Buffer.

SplSetQueue Parameter - ulParmNum

ulParmNum (**ULONG**) - input
Parameter number.

Specifies either that the entire **PRQINFO3** or **PRQINFO6** structure is to be modified, or that only one specific parameter is to be modified.

Possible values for this parameter are as follows:

Parameter	Constant (Value)
<i>uPriority</i>	PRQ_PRIORITY_PARMNUM (2)
<i>uStartTime</i>	PRQ_STARTTIME_PARMNUM (3)
<i>uUntilTime</i>	PRQ_UNTILTIME_PARMNUM (4)
<i>pszSepFile</i>	PRQ_SEPARATOR_PARMNUM (5)
<i>pszPrProc</i>	PRQ_PROCESSOR_PARMNUM (6)
<i>pszParms</i>	PRQ_PARS_PARMNUM (8)
<i>pszComment</i>	PRQ_COMMENT_PARMNUM (9)
<i>fsType</i>	PRQ_TYPE_PARMNUM (10)
<i>pszPrinters</i>	PRQ_PRINTERS_PARMNUM (12)
<i>pszDriverName</i>	PRQ_DRIVERNAME_PARMNUM (13)
<i>pDriverData</i>	PRQ_DRIVERDATA_PARMNUM (14)
<i>pszRemoteComputerName</i>	PRQ_REMOTE_COMPUTER_PARMNUM (15)
<i>pszRemoteQueueName</i>	PRQ_REMOTE_QUEUE_PARMNUM (16)

SplSetQueue Return Value - rc

rc (**SPLERR**) - returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
NERR_NetNotStarted (2102)	The network program is not installed.
NERR_RedirectedPath (2117)	The operation is invalid on a redirected resource.
NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_QNotFound (2150)	The printer queue does not exist.
NERR_DestNotFound (2152)	

The printer destination cannot be found.
NERR_DestNoRoom (2157)
The maximum number of printer destinations has been reached.
NERR_DestInvalidState (2162)
This operation cannot be performed on the print destination.
NERR_SpoolNoMemory (2165)
A spooler memory allocation failure occurred.
NERR_DriverNotFound (2166)
The device driver does not exist.
NERR_DataTypeInvalid (2167)
The datatype is not supported by the processor.
NERR_ProcNotFound (2168)
The queue processor is not installed.
NERR_BadDev (2341)
The requested device is invalid.
NERR_CommDevInUse (2343)
The requested device is invalid.
NERR_InvalidComputer (2351)
The computer name is invalid.

SplSetQueue - Parameters

pszComputerName (PSZ) - input
Name of computer where queue is to be modified.

A NULL string specifies the local workstation.

pszQueueName (PSZ) - input
Queue name.

ulLevel (ULONG) - input
Level of detail required.

This must be 3 or 6.

pBuf (PVOID) - input
Buffer.

If *ulParmNum* is 0, this parameter must contain a complete **PRQINFO3** or **PRQINFO6** structure. Otherwise, it must contain a valid new value for the parameter of the **PRQINFO3** or the **PRQINFO6** structure indicated in *ulParmNum*.

cbBuf (ULONG) - input
Size, in bytes, of Buffer.

ulParmNum (ULONG) - input
Parameter number.

Specifies either that the entire **PRQINFO3** or **PRQINFO6** structure is to be modified, or that only one specific parameter is to be modified.

Possible values for this parameter are as follows:

Parameter	Constant (Value)
<i>uPriority</i>	PRQ_PRIORITY_PARMNUM (2)
<i>uStartTime</i>	PRQ_STARTTIME_PARMNUM (3)
<i>uUntilTime</i>	PRQ_UNTILTIME_PARMNUM (4)
<i>pszSepFile</i>	PRQ_SEPARATOR_PARMNUM (5)
<i>pszPrProc</i>	PRQ_PROCESSOR_PARMNUM (6)
<i>pszParams</i>	PRQ_PARAMS_PARMNUM (8)
<i>pszComment</i>	PRQ_COMMENT_PARMNUM (9)
<i>fsType</i>	PRQ_TYPE_PARMNUM (10)
<i>pszPrinters</i>	PRQ_PRINTERS_PARMNUM (12)
<i>pszDriverName</i>	PRQ_DRIVERNAME_PARMNUM (13)
<i>pDriverData</i>	PRQ_DRIVERDATA_PARMNUM (14)

pszRemoteComputerName
pszRemoteQueueName

PRQ_REMOTE_COMPUTER_PARMNUM (15)
PRQ_REMOTE_QUEUE_PARMNUM (16)

rc ([SPLERR](#)) - returns
Return code.

NO_ERROR (0)
No errors occurred.

ERROR_ACCESS_DENIED (5)
Access is denied.

ERROR_NOT_SUPPORTED (50)
This request is not supported by the network.

ERROR_BAD_NETPATH (53)
The network path cannot be located.

ERROR_INVALID_PARAMETER (87)
An invalid parameter is specified.

ERROR_INVALID_LEVEL (124)
The level parameter is invalid.

NERR_NetNotStarted (2102)
The network program is not installed.

NERR_RedirectedPath (2117)
The operation is invalid on a redirected resource.

NERR_BufTooSmall (2123)
The API return buffer is too small.

NERR_QNotFound (2150)
The printer queue does not exist.

NERR_DestNotFound (2152)
The printer destination cannot be found.

NERR_DestNoRoom (2157)
The maximum number of printer destinations has been reached.

NERR_DestInvalidState (2162)
This operation cannot be performed on the print destination.

NERR_SpoolNoMemory (2165)
A spooler memory allocation failure occurred.

NERR_DriverNotFound (2166)
The device driver does not exist.

NERR_DataTypeInvalid (2167)
The datatype is not supported by the processor.

NERR_ProcNotFound (2168)
The queue processor is not installed.

NERR_BadDev (2341)
The requested device is invalid.

NERR_CommDevInUse (2343)
The requested device is invalid.

NERR_InvalidComputer (2351)
The computer name is invalid.

SplSetQueue - Remarks

If the *uPriority* field in [PRQINFO3](#) or [PRQINFO6](#) is set to PRQ_NO_PRIORITY, the queue priority is not changed.

SplSetQueue - Related Functions

Related Functions

- [SplCreateQueue](#)
- [SplEnumDevice](#)
- [SplEnumDriver](#)
- [SplEnumQueue](#)
- [SplEnumQueueProcessor](#)
- [SplQueryQueue](#)

SplSetQueue - Example Code

This sample code prompts the user to enter a parameter number and a value at the prompt. This value is then put into a buffer for use by the function.

```
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>      /* for printf function */
#include <stdlib.h>     /* for atoi function  */
#include <string.h>     /* for strlen function */

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    CHAR    bufValue[2] = {0};
    CHAR    bufInput[128]= {0};
    ULONG   splerr ;
    ULONG   cbBuf ;
    ULONG   ulParmNum ;
    USHORT  usParm ;
    PSZ     pszComputerName ;
    PSZ     pszQueueName ;
    PVOID    pBuf;

    if (argc != 2)
    {
        printf("Syntax:  setgryq QueueName \n");
        DosExit( EXIT_PROCESS , 0 ) ;
    }

    /* This function will be for the local workstation.
    pszComputerName = (PSZ)NULL ;

    /* Get the parameter from the command line.
    pszQueueName = argv[1];

    /* Prompt the user for the parameter and values, and put them in buffers. */
    printf("Enter Parameter number to be modified\n");
    gets(&bufValue[0]);
    printf("Enter new parameter value \n");
    gets(&bufInput[0]);

    /* Convert the ParmNum to a ULONG.
    ulParmNum = atoi(&bufValue[0]);
    switch (ulParmNum){
        case 2:
        case 3:
        case 4:
        case 10:
            /* Determine the size of the buffer needed.
            cbBuf = sizeof(PUSHORT);

            /* Convert the buffer input to a USHORT.
            usParm =(USHORT)atoi(&bufInput[0]);

            /* Set the pBuf pointer to point to the value obtained.
            pBuf = &usParm;
            break;
        case 5:
        case 6:
        case 8:
        case 9:
        case 12:
        case 13:
            /* Determine the size of the buffer needed.
            cbBuf = strlen(&bufInput[0])+1;
```

```

        /* Set the pBuf pointer to point to the value obtained from input. */
        pBuf = (PSZ)&bufInput;
        break;
    case 14:
        printf("For simplicity this is not implemented.");
        break;
    default:
        printf("Invalid number\n");
        DosExit( EXIT_PROCESS , 0 ) ;
        break;
}

/* Make the call with all the proper parameters. */
splerr = SplSetQueue(pszComputerName, pszQueueName, 3L,
                    pBuf, cbBuf, ulParmNum) ;

/* Print the resultant error code, and the parameters entered. */
printf("SplSetQueue Error= %ld, Parameter= %d, cbBuf= %ld,
        ulParmNum= %ld\n",
        splerr, usParm, cbBuf, ulParmNum);
DosExit( EXIT_PROCESS , 0 ) ;
return (splerr);
}

```

SplSetQueue - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

SplUnRegister

SplUnRegister - Syntax

Description:

This function allows applications to unregister for notification of events about printers. This function should be used when an application no longer wants certain alerts, in order to limit the network traffic.

```

#define INCL_SPL
#define INCL_SPLBIDI
#include <os2.h>

PSZ      pszComputerName; /* Printer server name to unregister for alert notification. */
PSZ      pszName;         /* Case-sensitive name of printer port to unregister for alerts. */
ULONG    ulHandle;        /* Spooler alert handle previously allocated by SplRegister. */

```

```

ULONG    ulCategory;        /* Category of alert to unregister. */
ULONG    ulType;            /* Type of alert to unregister for the given category. */
ULONG    rc;                /* Return codes. */

rc = SplUnRegister(pszComputerName, pszName,
    ulHandle, ulCategory, ulType);

```

SplUnRegister Parameter - pszComputerName

pszComputerName ([PSZ](#)) - input
 Printer server name to unregister for alert notification.

This value is NULL for local printer ports.

SplUnRegister Parameter - pszName

pszName ([PSZ](#)) - input
 Case-sensitive name of printer port to unregister for alerts.

This value can be a virtual port name.

SplUnRegister Parameter - ulHandle

ulHandle ([ULONG](#)) - input
 Spooler alert handle previously allocated by SplRegister.

SplUnRegister Parameter - ulCategory

ulCategory ([ULONG](#)) - input
 Category of alert to unregister.

This value can be the core alert category or an extended alert category as defined below:

PRTALERT_CATEGORY_ALL - 0x00000000

Unregister for all alerts for the given alert handle. This will remove registration for all printer ports (even remote ones) for the alert handle.

PRTALERT_CATEGORY_ALL_CORE - 0x00000002

Unregister for all core alerts for the port.

PRTALERT_CATEGORY_ALL_EXT - 0x00000003

Unregister for all extended alerts for the port.

PRTALERT_CATEGORY_CORE - 0x00000001

Core alert category. This will remove registration for only those core alerts whose bits are set in *ulType* for the port.

PRTALERT_CATEGORY_EXT_MAX - 0x0000DFFF

End of the extended alert categories.

PRTALERT_CATEGORY_EXTENDED - 0x0000D000

Beginning of the extended alert categories. Protocol converters should limit the category range to between 0x0000D000 and 0x0000DFFF.

If the protocol converter does not support the given category, ERROR_INVALID_CATEGORY(117) is returned.

SplUnRegister Parameter - ulType

ulType (ULONG) - input

Type of alert to unregister for the given category.

If the category is PRTALERT_CATEGORY_CORE, then this is a bit-field that defines the core alerts to unregister.

If the category is **not** PRTALERT_CATEGORY_ALL, PRTALERT_CATEGORY_CORE, PRTALERT_CATEGORY_ALL_CORE, or PRTALERT_CATEGORY_ALL_EXT, this is a value that defines the specific extended alert to unregister.

SplUnRegister Return Value - rc

rc (ULONG) - returns

Return codes.

0 Success

ERROR_FILE_NOT_FOUND(2)
pszName is not a valid port name.

ERROR_INVALID_HANDLE(6)
The value *ulHandle* is not a valid spooler alert handle.

ERROR_INVALID_PARAMETER(87)
An invalid buffer given.

ERROR_NOT_SUPPORTED(50)
The spooler on the given print server cannot support alert notification.

PMERR_SPL_NOT_REGISTERED(0x4042)
The alert handle *ulHandle* is not currently registered for any alerts specified by *ulCategory* and *ulType*.

PMERR_SPL_SPOOLER_NOT_INSTALLED(0x4009)
The spooler is not enabled.

SplUnRegister - Parameters

pszComputerName (PSZ) - input
Printer server name to unregister for alert notification.

This value is NULL for local printer ports.

pszName (PSZ) - input
Case-sensitive name of printer port to unregister for alerts.

This value can be a virtual port name.

ulHandle (ULONG) - input
Spooler alert handle previously allocated by SplRegister.

ulCategory (ULONG) - input
Category of alert to unregister.

This value can be the core alert category or an extended alert category as defined below:

PRTALERT_CATEGORY_ALL - 0x00000000
Unregister for all alerts for the given alert handle. This will remove registration for all printer ports (even remote ones) for the alert handle.

PRTALERT_CATEGORY_ALL_CORE - 0x00000002
Unregister for all core alerts for the port.

PRTALERT_CATEGORY_ALL_EXT - 0x00000003
Unregister for all extended alerts for the port.

PRTALERT_CATEGORY_CORE - 0x00000001
Core alert category. This will remove registration for only those core alerts whose bits are set in *ulType* for the port.

PRTALERT_CATEGORY_EXT_MAX - 0x0000DFFF
End of the extended alert categories.

PRTALERT_CATEGORY_EXTENDED - 0x0000D000
Beginning of the extended alert categories. Protocol converters should limit the category range to between 0x0000D000 and 0x0000DFFF.

If the protocol converter does not support the given category, ERROR_INVALID_CATEGORY(117) is returned.

ulType (ULONG) - input
Type of alert to unregister for the given category.

If the category is PRTALERT_CATEGORY_CORE, then this is a bit-field that defines the core alerts to unregister.

If the category is **not** PRTALERT_CATEGORY_ALL, PRTALERT_CATEGORY_CORE, PRTALERT_CATEGORY_ALL_CORE, or PRTALERT_CATEGORY_ALL_EXT, this is a value that defines the specific extended alert to unregister.

rc (ULONG) - returns
Return codes.

0 Success

ERROR_FILE_NOT_FOUND(2)
pszName is not a valid port name.

ERROR_INVALID_HANDLE(6)
The value *ulHandle* is not a valid spooler alert handle.

ERROR_INVALID_PARAMETER(87)
An invalid buffer given.

ERROR_NOT_SUPPORTED(50)
The spooler on the given print server cannot support alert notification.

PMERR_SPL_NOT_REGISTERED(0x4042)
The alert handle *ulHandle* is not currently registered for any alerts specified by *ulCategory* and *ulType*.

PMERR_SPL_SPOOLER_NOT_INSTALLED(0x4009)
The spooler is not enabled.

SplUnRegister - Remarks

Registration is **not** maintained across reboots.

SplUnRegister - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

Window Functions

The PM user interface is based on *windows*-areas of the screen through which applications interact with the user. This section contains an alphabetical list of the functions which are available to the application for using and controlling windows.

WinAddAtom

WinAddAtom - Syntax

This function adds an atom to an atom table.

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HATOMTBL    hatomtblAtomTbl; /* Atom-table handle. */
PSZ         AtomName;        /* Atom name. */
ATOM        atom;             /* Atom value. */

atom = WinAddAtom(hatomtblAtomTbl, AtomName);
```

WinAddAtom Parameter - hatomtblAtomTbl

hatomtblAtomTbl ([HATOMTBL](#)) - input
Atom-table handle.

This is the handle returned by a previous call to [WinCreateAtomTable](#) or [WinQuerySystemAtomTable](#).

WinAddAtom Parameter - AtomName

AtomName ([PSZ](#)) - input
Atom name.

This is a character string to be added to the table.

If the string begins with an "#" character, the five ASCII digits that follow are converted into an integer atom. If this integer is a valid integer atom, this function returns that atom, without modifying the atom table.

If the string begins with an "!" character, the next four bytes are interpreted as an atom. If it is an integer atom, that atom is returned. If it is not an integer atom and it is a valid atom for the given atom table (that is, it has an atom name and use count associated with it) the use count of that atom is incremented by one and the atom is returned. Otherwise 0 is returned.

If the high order word of the string is minus one, the low order word is an atom. If it is an integer atom, that atom is returned. If it is not

an integer atom and it is a valid atom for the given atom table (that is, it has an atom name and use count associated with it) the use count of that atom is incremented by one and the atom is returned. Otherwise 0 is returned.

WinAddAtom Return Value - atom

atom ([ATOM](#)) - returns
Atom value.

Atom

The atom associated with the passed string

0

Invalid atom-table handle or invalid atom name specified.

WinAddAtom - Parameters

hatomtblAtomTbl ([HATOMTBL](#)) - input
Atom-table handle.

This is the handle returned by a previous call to [WinCreateAtomTable](#) or [WinQuerySystemAtomTable](#).

AtomName ([PSZ](#)) - input
Atom name.

This is a character string to be added to the table.

If the string begins with an "#" character, the five ASCII digits that follow are converted into an integer atom. If this integer is a valid integer atom, this function returns that atom, without modifying the atom table.

If the string begins with an "!" character, the next four bytes are interpreted as an atom. If it is an integer atom, that atom is returned. If it is not an integer atom and it is a valid atom for the given atom table (that is, it has an atom name and use count associated with it) the use count of that atom is incremented by one and the atom is returned. Otherwise 0 is returned.

If the high order word of the string is minus one, the low order word is an atom. If it is an integer atom, that atom is returned. If it is not an integer atom and it is a valid atom for the given atom table (that is, it has an atom name and use count associated with it) the use count of that atom is incremented by one and the atom is returned. Otherwise 0 is returned.

atom ([ATOM](#)) - returns
Atom value.

Atom

The atom associated with the passed string

0

Invalid atom-table handle or invalid atom name specified.

WinAddAtom - Remarks

If the atom name represents an integer atom, this function returns the atom represented by the passed atom name.

If the atom name does not represent an integer atom and if the atom name already exists in the atom table, this function increments the use count of that atom by one. Otherwise, the atom is added to the table and its use count is set to one. In either case this function returns the atom represented by the passed atom name.

WinAddAtom - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HATOMTBL (0x1013)

An invalid atom-table handle was specified.

PMERR_INVALID_INTEGER_ATOM (0x1016)

The specified atom is not a valid integer atom.

PMERR_INVALID_ATOM_NAME (0x1015)

An invalid atom name string was passed.

PMERR_ATOM_NAME_NOT_FOUND (0x1017)

The specified atom name is not in the atom table.

WinAddAtom - Related Functions

Related Functions

- [WinAddAtom](#)
 - [WinCreateAtomTable](#)
 - [WinDeleteAtom](#)
 - [WinDestroyAtomTable](#)
 - [WinFindAtom](#)
 - [WinQueryAtomLength](#)
 - [WinQueryAtomName](#)
 - [WinQueryAtomUsage](#)
 - [WinQuerySystemAtomTable](#)
-

WinAddAtom - Example Code

This example creates an Atom Table and then adds the atom "newatom" to the new table; it then checks the count for this new atom to verify that it is 1.

```
#define INCL_WINATOM          /* Window Atom Functions          */
#include <os2.h>

ATOM  atom;                  /* new atom value          */
HATOMTBL  hatomtblAtomTbl; /* atom-table handle      */
char  pszAtomName[10]; /* atom name              */
ULONG  ulInitial = 0; /* initial atom table size (use default) */
ULONG  ulBuckets = 0; /* size of hash table (use default) */
ULONG  ulCount; /* atom usage count      */
BOOL  atomCount1 = FALSE; /* indicates atom count != 1 */

/* create atom table of default size */
hatomtblAtomTbl = WinCreateAtomTable(ulInitial, ulBuckets);

/* define name for new atom and add to table */
strcpy(pszAtomName, "newatom");
atom = WinAddAtom(hatomtblAtomTbl, pszAtomName);

ulCount = WinQueryAtomUsage(hatomtblAtomTbl, atom);

/* verify that usage count is 1 */
if (ulCount == 1)
```

```
atomCount1 = TRUE;
```

WinAddAtom - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinAddSwitchEntry

WinAddSwitchEntry - Syntax

This function adds an entry to the Window List. This is a list of running programs that is displayed to the user by the operating system.

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

SWCNTRL    swctlSwitchData; /* Switch data. */
HSWITCH    hswitchSwitch; /* Handle to the newly created Window List entry. */

hswitchSwitch = WinAddSwitchEntry(swctlSwitchData);
```

WinAddSwitchEntry Parameter - swctlSwitchData

swctlSwitchData ([SWCNTRL](#)) - input
Switch data.

Contains information about the newly created Window List entry.

If the *szSwtitle* field of the [SWCNTRL](#) structure is 0, the system uses the name under which the application is started. This only applies for the first call to this function since the program started. Otherwise, a NULL entry name is invalid.

The title is truncated, if necessary, to 60 characters.

If the *hprog* field of the [SWCNTRL](#) structure is NULLHANDLE, the value used by the system when the program was loaded (if it has been loaded) is substituted.

If the *idProcess* field of the [SWCCTRL](#) structure is 0, the current process ID is used.

If the *idSession* field of the [SWCCTRL](#) structure is 0, the current session ID is used.

If the *hwndIcon* field of the [SWCCTRL](#) structure is NULLHANDLE, the system supplies a default icon.

WinAddSwitchEntry Return Value - hswitchSwitch

hswitchSwitch ([HSWITCH](#)) - returns

Handle to the newly created Window List entry.

There is a system limit to the number of Window List entries. However, this is a large number (several hundred) and is unlikely to be reached in practice because other system limits, such as memory size, restrict the number before this limit is reached.

NULLHANDLE

Error occurred

Other

Handle to the newly created Window List entry.

WinAddSwitchEntry - Parameters

swctlSwitchData ([SWCCTRL](#)) - input

Switch data.

Contains information about the newly created Window List entry.

If the *szSwtitle* field of the [SWCCTRL](#) structure is 0, the system uses the name under which the application is started. This only applies for the first call to this function since the program started. Otherwise, a NULL entry name is invalid.

The title is truncated, if necessary, to 60 characters.

If the *hprog* field of the [SWCCTRL](#) structure is NULLHANDLE, the value used by the system when the program was loaded (if it has been loaded) is substituted.

If the *idProcess* field of the [SWCCTRL](#) structure is 0, the current process ID is used.

If the *idSession* field of the [SWCCTRL](#) structure is 0, the current session ID is used.

If the *hwndIcon* field of the [SWCCTRL](#) structure is NULLHANDLE, the system supplies a default icon.

hswitchSwitch ([HSWITCH](#)) - returns

Handle to the newly created Window List entry.

There is a system limit to the number of Window List entries. However, this is a large number (several hundred) and is unlikely to be reached in practice because other system limits, such as memory size, restrict the number before this limit is reached.

NULLHANDLE

Error occurred

Other

Handle to the newly created Window List entry.

WinAddSwitchEntry - Remarks

Neither this function nor the [WinRemoveSwitchEntry](#) function are required if the main window is created with the frame creation flags FCF_TASKLIST or FCF_STANDARD, because these flags automatically update the Window List when the main window is created or destroyed.

WinAddSwitchEntry - Errors

Possible returns from [WinGetLastError](#)

PMERR_NO_SPACE (0x1201)

The limit on the number of Window List entries has been reached with WinAddSwitchEntry.

PMERR_INVALID_SESSION_ID (0x120B)

The specified session identifier is invalid. Either zero (for the application's own session) or a valid identifier must be specified.

PMERR_INVALID_WINDOW (0x1206)

The window specified with a Window List call is not a valid frame window.

WinAddSwitchEntry - Related Functions

Related Functions

- [WinAddSwitchEntry](#)
- [WinChangeSwitchEntry](#)
- [WinCreateSwitchEntry](#)
- [WinQuerySessionTitle](#)
- [WinQuerySwitchEntry](#)
- [WinQuerySwitchHandle](#)
- [WinQuerySwitchList](#)
- [WinQueryTaskSizePos](#)
- [WinQueryTaskTitle](#)
- [WinRemoveSwitchEntry](#)
- [WinSwitchToProgram](#)

WinAddSwitchEntry - Example Code

This example calls WinQueryWindowProcess to get the current process identifier (needed for the SWCNTRL structure). It then sets up the swctl structure and calls WinAddSwitchEntry to add the name of the program to the task list. The returned handle can be used in subsequent calls to WinChangeSwitchEntry if the title needs to be changed. The variables swctl, hswitch, and pid should be global if the application will be calling the WinChangeSwitchEntry function to avoid having to set up the structure again.

```
#define INCL_WINSWITCHLIST          /* Window Task Switch functions */
#include <os2.h>

SWCNTRL swctl;                     /* Switch control data */
HSWITCH hswitch;                   /* Switch handle */
PID pid;                           /* Process id */
HWND hwndFrame;                   /* Frame handle */

WinQueryWindowProcess(hwndFrame, &pid, NULL);

swctl.hwnd = hwndFrame;             /* Window handle */
swctl.hwndIcon = NULLHANDLE;       /* Icon handle */
swctl.hprog = NULLHANDLE;          /* Program handle */
swctl.idProcess = pid;              /* Process identifier */
swctl.idSession = 0;               /* Session identifier */
```

```

swctl.uchVisibility = SWL_VISIBLE; /* Visibility          */
swctl.fbJump = SWL_JUMPABLE;      /* Jump indicator  */
swctl.szSwttitle[0] = 0;          /* Program name    */
swctl.bProgType = PROG_DEFAULT;   /* Program type    */

hswitch = WinAddSwitchEntry(&swctl);

```

WinAddSwitchEntry - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinAlarm

WinAlarm - Syntax

This function generates an audible alarm.

```

#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND    hwndDeskTop; /* Desktop-window handle. */
ULONG   flStyle;     /* Alarm style. */
BOOL     rc;          /* Alarm-generated indicator. */

rc = WinAlarm(hwndDeskTop, flStyle);

```

WinAlarm Parameter - hwndDeskTop

hwndDeskTop (HWND) - input
Desktop-window handle.

HWND_DESKTOP	The desktop window
Other	Specified desktop window.

WinAlarm Parameter - flStyle

flStyle ([ULONG](#)) - input
Alarm style.

Used to signify different situations to the operator.

The duration and frequency of the alarms can be changed by the [WinSetSysValue](#) function. The alarm frequency is defined to be in the range 0x0025 through 0x7FFF. The alarm is not generated if system value SV_ALARM is set to FALSE. The alarms are dependent on the device capability. Different alarms are selected by use of these values:

- WA_WARNING
- WA_NOTE
- WA_ERROR

WinAlarm Return Value - rc

rc ([BOOL](#)) - returns
Alarm-generated indicator.

- | | |
|-------|----------------------|
| TRUE | Alarm generated |
| FALSE | Alarm not generated. |

WinAlarm - Parameters

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

- | | |
|--------------|---------------------------|
| HWND_DESKTOP | The desktop window |
| Other | Specified desktop window. |

flStyle ([ULONG](#)) - input
Alarm style.

Used to signify different situations to the operator.

The duration and frequency of the alarms can be changed by the [WinSetSysValue](#) function. The alarm frequency is defined to be in the range 0x0025 through 0x7FFF. The alarm is not generated if system value SV_ALARM is set to FALSE. The alarms are dependent on the device capability. Different alarms are selected by use of these values:

- WA_WARNING
- WA_NOTE

WA_ERROR

rc ([BOOL](#)) - returns
Alarm-generated indicator.

TRUE	Alarm generated
FALSE	Alarm not generated.

WinAlarm - Remarks

Although this function is in the INCL_WINDIALOGS section, it is part of the common subset and is also included if INCL_COMMON is defined.

WinAlarm - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_INVALID_PARM (0x1303)
A parameter to the function contained invalid data.

WinAlarm - Related Functions

Related Functions

- [WinAlarm](#)
- [WinFlashWindow](#)
- [WinMessageBox](#)

WinAlarm - Example Code

This example calls an application-defined initialization function, and calls WinAlarm to generate an audible alarm to notify the user if the function fails.

```
#define INCL_WINDIALOGS          /* Window Dialog Mgr Functions */
#include <os2.h>

if (!GenericInit())              /* general initialization */
    WinAlarm(HWND_DESKTOP, WA_ERROR);
```


WinAlarm - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinAssociateHelpInstance

WinAssociateHelpInstance - Syntax

This function associates the specified instance of the Help Manager with the window chain of the specified application window.

```
#define INCL_WINHELP /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND     hwndHelpInstance; /* Handle of an instance of the Help Manager. */
HWND     hwndApp;          /* Handle of an application window. */
BOOL     rc;               /* Success indicator. */

rc = WinAssociateHelpInstance(hwndHelpInstance,
                              hwndApp);
```

WinAssociateHelpInstance Parameter - hwndHelpInstance

hwndHelpInstance ([HWND](#)) - input

Handle of an instance of the Help Manager.

This is the handle returned by the [WinCreateHelpInstance](#) call.

NULLHANDLE

Disassociates an instance of the Help Manager from a window chain when the instance has been destroyed.

Other

The handle of an instance of the Help Manager to be associated with the application window chain.

WinAssociateHelpInstance Parameter - hwndApp

hwndApp ([HWND](#)) - input
Handle of an application window.

The handle of the application window with which the instance of the Help Manager will be associated. The instance of the Help Manager is associated with the application window and any of its children or owned windows.

WinAssociateHelpInstance Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinAssociateHelpInstance - Parameters

hwndHelpInstance ([HWND](#)) - input
Handle of an instance of the Help Manager.

This is the handle returned by the [WinCreateHelpInstance](#) call.

NULLHANDLE	Disassociates an instance of the Help Manager from a window chain when the instance has been destroyed.
Other	The handle of an instance of the Help Manager to be associated with the application window chain.

hwndApp ([HWND](#)) - input
Handle of an application window.

The handle of the application window with which the instance of the Help Manager will be associated. The instance of the Help Manager is associated with the application window and any of its children or owned windows.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinAssociateHelpInstance - Remarks

In order to provide help, the application must associate an instance of the Help Manager with a chain of application windows. This association lets the Help Manager know which instance should provide the help function.

The Help Manager traces the window chain, starting from the window where help is requested. The application window in the chain with the

associated help instance will be the one with which the Help Manager communicates and next to which the help window is positioned, unless a [HM_SET_ACTIVE_WINDOW](#) message is sent to the Help Manager. If the [HM_SET_ACTIVE_WINDOW](#) message is sent to the Help Manager, the active window parameter is the window with which the Help Manager communicates. The Help Manager positions the help window next to the window specified as the relative window.

WinAssociateHelpInstance - Related Functions

Related Functions

- [WinAssociateHelpInstance](#)
 - [WinCreateHelpInstance](#)
 - [WinCreateHelpTable](#)
 - [WinDestroyHelpInstance](#)
 - [WinLoadHelpTable](#)
 - [WinQueryHelpInstance](#)
-

WinAssociateHelpInstance - Related Messages

Related Messages

- [HM_SET_ACTIVE_WINDOW](#)
-

WinAssociateHelpInstance - Example Code

This example shows a typical main function for an application which uses help. Following creation of the main application window the Help Manager is initialized and associated with the window. The help table is defined in the application's resources. When the window is destroyed, terminating the application, the help instance is also destroyed.

```
#define INCL=_WIN
#include <os2.h>

#define IDHT_APPLICATION      100      /* id of HELP TABLE in resource file */

main( int argc, char *argv[], char *envp[] )
{
    HAB  hab = WinInitialize( 0 );
    HMQ  hmq = WinCreateMsgQueue( hab, 0 );
    HWND hwnd;
    HWND hwndClient;
    HWND hwndHelp;
    QMSG qmsg;
    ULONG flStyle;
    HELPINIT helpinit;

    /* Setup the help initialization structure */
    helpinit.cb = sizeof( HELPINIT );
    helpinit.ulReturnCode = 0L;
    helpinit.pszTutorialName = (PSZ)NULL;
    /* Help table in application resource */
    helpinit.phtHelpTable = (PHELPTABLE)MAKEULONG( IDHT_APPLICATION, 0xffff );
    helpinit.hmodHelpTableModule = NULLHANDLE;
    /* Default action bar and accelerators */
    helpinit.hmodAccelActionBarModule = NULLHANDLE;
    helpinit.idAccelTable = 0;
    helpinit.idActionBar = 0;
    helpinit.pszHelpWindowTitle = "APPNAME HELP";
    helpinit.fShowPanelId = CMIC_SHOW_PANEL_ID;
```

```

helpinit.pszHelpLibraryName = "APPNAME.HLP";

/* Register the class */
if( WinRegisterClass( ... ) )
{
    /* create the main window */
    flStyle = FCF_STANDARD;
    hwnd = WinCreateStdWindow( ... );

    if( hwnd )
    {
        /* Create and associate the help instance */
        hwndHelp = WinCreateHelpInstance( hab, &helpinit );

        if( hwndHelp && WinAssociateHelpInstance( hwndHelp, hwnd ) )
        {
            /* Process messages */
            while( WinGetMsg( hab, &qmsg, NULLHANDLE, 0, 0 ) )
            {
                WinDispatchMsg( hab, &qmsg );
            } /* endwhile */
        }

        /* Remove help instance - note: add */
        /*      WinAssociateHelpInstance( NULLHANDLE, hwnd ); */
        /* to WM_DESTROY processing to remove the association. */
        WinDestroyHelpInstance( hwndHelp );
    }
}

/* finish the cleanup and exit */
WinDestroyMsgQueue( hmq );
WinTerminate( hab );
}

```

WinAssociateHelpInstance - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinBeginEnumWindows

WinBeginEnumWindows - Syntax

This function begins the enumeration process for all of the immediate child windows of a specified window.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
```

```
#include <os2.h>

HWND    hwnd;          /* Handle of the window whose child windows are to be enumerated. */
HENUM    henumHenum;    /* Enumeration handle. */

henumHenum = WinBeginEnumWindows(hwnd);
```

WinBeginEnumWindows Parameter - hwnd

hwnd ([HWND](#)) - input

Handle of the window whose child windows are to be enumerated.

HWND_DESKTOP

Enumerate all main windows

HWND_OBJECT

Enumerate all object windows

Other

Enumerate all immediate children of the specified window.

WinBeginEnumWindows Return Value - henumHenum

henumHenum ([HENUM](#)) - returns

Enumeration handle.

This is used in subsequent calls to the [WinGetNextWindow](#) function to return the immediate child-window handles in succession.

When the application has finished the enumeration, the enumeration handle must be destroyed with the [WinEndEnumWindows](#) call.

WinBeginEnumWindows - Parameters

hwnd ([HWND](#)) - input

Handle of the window whose child windows are to be enumerated.

HWND_DESKTOP

Enumerate all main windows

HWND_OBJECT

Enumerate all object windows

Other

Enumerate all immediate children of the specified window.

henumHenum ([HENUM](#)) - returns

Enumeration handle.

This is used in subsequent calls to the [WinGetNextWindow](#) function to return the immediate child-window handles in succession.

When the application has finished the enumeration, the enumeration handle must be destroyed with the [WinEndEnumWindows](#) call.

WinBeginEnumWindows - Remarks

This function remembers the window hierarchy at the time of invocation of the call. Thereafter the information is referenced by use of the *henum/henum* parameter and does not change during the enumeration by the [WinGetNextWindow](#) call. The windows are enumerated in the z-order at the time enumeration is begun, with the topmost child window enumerated first.

Only the immediate children of the specified window are enumerated; child windows of the child windows are excluded.

The enumerated windows are not locked by this function and can thus be destroyed between the time that it is called and the time that the [WinGetNextWindow](#) function is used to obtain the handle for the window.

WinBeginEnumWindows - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinBeginEnumWindows - Related Functions

Related Functions

- [WinBeginEnumWindows](#)
- [WinEndEnumWindows](#)
- [WinEnumDlgItem](#)
- [WinGetNextWindow](#)
- [WinIsChild](#)
- [WinMultWindowFromIDs](#)
- [WinQueryWindow](#)
- [WinSetOwner](#)
- [WinSetParent](#)

WinBeginEnumWindows - Example Code

This example begins window enumeration of all main windows (i.e. all immediate children of the Desktop), after which [WinGetNextWindow](#) is called in a loop to enumerate all the children, until all children are found and the enumeration ends.

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#include <os2.h>

HWND  hwndParent;             /* Handle of the window whose child windows
                                are to be enumerated */
HWND  hwndNext;               /* current enumeration handle */
HENUM henum;                  /* enumeration handle */
BOOL  fSuccess;               /* success indicator */
SHORT sRetLen;                /* returned string length */
SHORT sLength = 10;           /* string buffer length */
char  pchBuffer[10];          /* string buffer */

hwndParent = HWND_DESKTOP;

henum = WinBeginEnumWindows(hwndParent);
```

```

while ((hwndNext = WinGetNextWindow(henum)) != NULLHANDLE) {
    .
    .
}

fSuccess = WinEndEnumWindows (henum);

```

WinBeginEnumWindows - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinBeginPaint

WinBeginPaint - Syntax

This function obtains a presentation space whose associated update region is set ready for drawing in a specified window.

```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND      hwnd;          /* Handle of window where drawing is going to occur. */
HPS       hps;           /* Presentation-space handle. */
PRECTL    prclPaint;     /* Bounding rectangle. */
HPS       hpsPaintPS;    /* Presentation-space handle. */

hpsPaintPS = WinBeginPaint(hwnd, hps, prclPaint);

```

WinBeginPaint Parameter - hwnd

hwnd ([HWND](#)) - input

Handle of window where drawing is going to occur.

HWND_DESKTOP

The desk top window.

Other

Specified window.

WinBeginPaint Parameter - hps

hps ([HPS](#)) - input

Presentation-space handle.

NULLHANDLE

Obtain a cache presentation space.

Other

Presentation-space handle. This function sets its clipping region to the update region of the *hwnd* parameter.

WinBeginPaint Parameter - prclPaint

prclPaint ([PRECTL](#)) - output

Bounding rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

NULL

No bounding rectangle; that is, there is no need of changing any point.

Other

Specifies the smallest rectangle bounding the update region, in window coordinates.

WinBeginPaint Return Value - hpsPaintPS

hpsPaintPS ([HPS](#)) - returns

Presentation-space handle.

NULLHANDLE

Error occurred

Other

Presentation-space handle.

WinBeginPaint - Parameters

hwnd ([HWND](#)) - input

Handle of window where drawing is going to occur.

HWND_DESKTOP

The desk top window.

Other

Specified window.

hps ([HPS](#)) - input

Presentation-space handle.

NULLHANDLE

Obtain a cache presentation space.

Other

Presentation-space handle. This function sets its clipping region to the update region of the *hwnd* parameter.

prclPaint ([PRECTL](#)) - output

Bounding rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

NULL

No bounding rectangle; that is, there is no need of changing any point.

Other

Specifies the smallest rectangle bounding the update region, in window coordinates.

hpsPaintPS ([HPS](#)) - returns

Presentation-space handle.

NULLHANDLE

Error occurred

Other

Presentation-space handle.

WinBeginPaint - Remarks

This function is generally call during the processing of a [WM_PAINT](#) message when the application needs to update the content of the window.

If the presentation space already exists, its update region is set and the device context of the window is associated with the presentation space. Otherwise, a cache presentation space is obtained specifically for the window.

The update region associated with *hwnd* is reset to NULLHANDLE. It is assumed that any drawing following this function restores the content of the window to a fully correct state.

This function hides the pointer if it is in the window and the [WinEndPaint](#) function restores it.

This function hides the tracking rectangle if it is active and might hide part of the painting window; that is, if *hwnd* is a child of the window specified by the *hwnd* parameter in the [WinTrackRect](#) function. The [WinEndPaint](#) function shows it again.

[WinEndPaint](#) must be called after the application completes drawing, and can be nested for the same window.

WinBeginPaint - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)

An invalid window handle was specified.

PMERR_INV_HPS (0x207F)

An invalid presentation-space handle was specified.

WinBeginPaint - Related Functions

Related Functions

- [WinBeginPaint](#)
- [WinEnableWindowUpdate](#)
- [WinEndPaint](#)
- [WinExcludeUpdateRegion](#)
- [WinGetClipPS](#)
- [WinGetPS](#)
- [WinGetScreenPS](#)
- [WinInvalidateRect](#)
- [WinInvalidateRegion](#)
- [WinIsWindowShowing](#)
- [WinIsWindowVisible](#)
- [WinLockVisRegions](#)
- [WinOpenWindowDC](#)
- [WinQueryUpdateRect](#)
- [WinQueryUpdateRegion](#)
- [WinRealizePalette](#)
- [WinReleasePS](#)
- [WinShowWindow](#)
- [WinUpdateWindow](#)
- [WinValidateRect](#)
- [WinValidateRegion](#)

WinBeginPaint - Related Messages

Related Messages

- [WM_PAINT](#)

WinBeginPaint - Example Code

This example uses WinBeginPaint to obtain and associate a presentation space with the update region of a window so that redrawing can take place.

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#include <os2.h>

HWND    hwnd;                /* parent window */
RECT    rcl;                 /* update region */
HPS     hps;                 /* presentation-space handle */

case WM_PAINT:
    hps = WinBeginPaint(hwnd, /* handle of the window */
        NULLHANDLE,          /* get a cache presentation space */
        &rcl);               /* receives update rectangle */
    WinFillRect(hps, &rcl, CLR_WHITE);
    WinEndPaint(hps);
```

WinBeginPaint - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinBroadcastMsg

WinBroadcastMsg - Syntax

This function broadcasts a message to multiple windows.

```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndParent; /* Parent-window handle. */
ULONG     ulMsgId;     /* Message identifier. */
MPARAM    mpParam1;    /* Parameter 1. */
MPARAM    mpParam2;    /* Parameter 2. */
ULONG     flCmd;       /* Broadcast message command. */
BOOL      rc;          /* Success indicator. */

rc = WinBroadcastMsg(hwndParent, ulMsgId,
                    mpParam1, mpParam2, flCmd);
```

WinBroadcastMsg Parameter - hwndParent

hwndParent ([HWND](#)) - input
Parent-window handle.

WinBroadcastMsg Parameter - ulMsgId

ulMsgId ([ULONG](#)) - input
Message identifier.

WinBroadcastMsg Parameter - mpParam1

mpParam1 (MPARAM) - input
Parameter 1.

WinBroadcastMsg Parameter - mpParam2

mpParam2 (MPARAM) - input
Parameter 2.

WinBroadcastMsg Parameter - flCmd

flCmd (ULONG) - input
Broadcast message command.

- BMSG_POST
Post the message. This value is mutually exclusive with BMSG_SEND and BMSG_POSTQUEUE.
- BMSG_SEND
Send the message. This value is mutually exclusive with BMSG_POST and BMSG_POSTQUEUE.
- BMSG_POSTQUEUE
Post a message to all threads that have a message queue. This value is mutually exclusive with BMSG_POST and BMSG_SEND. The *hwnd* parameter of the QMSG structure is set to NULL.
- BMSG_DESCENDANTS
Broadcast the message to all the descendants of the *hwndParent* parameter.
- BMSG_FRAMEONLY
Broadcast the message only to windows with a style of CS_FRAME.

WinBroadcastMsg Return Value - rc

rc (BOOL) - returns
Success indicator.

- TRUE
Message was sent or posted successfully to all applicable windows
- FALSE
Error occurred.

WinBroadcastMsg - Parameters

hwndParent ([HWND](#)) - input
Parent-window handle.

ulMsgId ([ULONG](#)) - input
Message identifier.

mpParam1 ([MPARAM](#)) - input
Parameter 1.

mpParam2 ([MPARAM](#)) - input
Parameter 2.

flCmd ([ULONG](#)) - input
Broadcast message command.

BMSG_POST
Post the message. This value is mutually exclusive with **BMSG_SEND** and **BMSG_POSTQUEUE**.

BMSG_SEND
Send the message. This value is mutually exclusive with **BMSG_POST** and **BMSG_POSTQUEUE**.

BMSG_POSTQUEUE
Post a message to all threads that have a message queue. This value is mutually exclusive with **BMSG_POST** and **BMSG_SEND**. The *hwnd* parameter of the [QMSG](#) structure is set to NULL.

BMSG_DESCENDANTS
Broadcast the message to all the descendants of the *hwndParent* parameter.

BMSG_FRAMEONLY
Broadcast the message only to windows with a style of **CS_FRAME**.

rc ([BOOL](#)) - returns
Success indicator.

TRUE
Message was sent or posted successfully to all applicable windows

FALSE
Error occurred.

WinBroadcastMsg - Remarks

This function sends or posts a message to all the immediate child windows of *hwndParent*, except in the case when *flCmd* is **BMSG_DESCENDANTS**.

The *ulMsgId*, *mpParam1*, and *mpParam2* parameters make up the message sent or posted. The window handle of the receiving window is added to the message.

WinBroadcastMsg - Related Functions

Related Functions

- [WinBroadcastMsg](#)
- [WinCreateMsgQueue](#)
- [WinDestroyMsgQueue](#)
- [WinDispatchMsg](#)
- [WinGetDlgMsg](#)
- [WinGetMsg](#)
- [WinInSendMessage](#)

- [WinPeekMsg](#)
- [WinPostMsg](#)
- [WinPostQueueMsg](#)
- [WinQueryMsgPos](#)
- [WinQueryMsgTime](#)
- [WinQueryQueueInfo](#)
- [WinQueryQueueStatus](#)
- [WinSendDlgItemMsg](#)
- [WinSendMsg](#)
- [WinSetClassMsgInterest](#)
- [WinSetMsgInterest](#)
- [WinSetMsgMode](#)
- [WinSetSynchroMode](#)
- [WinWaitMsg](#)

WinBroadcastMsg - Example Code

This example broadcasts a WM_CLOSE message to all descendants of the specified window.

```
#define INCL_WINMESSAGEGR      /* Window Message Functions */
#include <os2.h>

BOOL fSuccess;                /* Success indicator */
HWND hwndParent;              /* parent window handle */
ULONG ulMsgId;                /* Message identifier */
MPARAM mpParam1;              /* Parameter 1 */
MPARAM mpParam2;              /* Parameter 2 */
ULONG flCmd;                  /* message command */

/* set msg to close window, parameters to NULL */
ulMsgId = WM_CLOSE;
mpParam1 = MPVOID;
mpParam2 = MPVOID;

/* broadcast to all descendants */
flCmd = BMSG_DESCENDANTS;

fSuccess = WinBroadcastMsg(hwndParent, ulMsgId, mpParam1,
                           mpParam2, flCmd);
```

WinBroadcastMsg - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinCalcFrameRect

WinCalcFrameRect - Syntax

This function calculates a client rectangle from a frame rectangle, or a frame rectangle from a client rectangle.

```
#define INCL_WINFRAMEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND     hwndFrame; /* Frame-window handle. */
PRECTL   prcl;      /* Window rectangle. */
BOOL     fClient;   /* Frame indicator. */
BOOL     rc;         /* Rectangle-calculated indicator. */

rc = WinCalcFrameRect(hwndFrame, prcl, fClient);
```

WinCalcFrameRect Parameter - hwndFrame

hwndFrame ([HWND](#)) - input
Frame-window handle.

WinCalcFrameRect Parameter - prcl

prcl ([PRECTL](#)) - in/out
Window rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

WinCalcFrameRect Parameter - fClient

fClient ([BOOL](#)) - input
Frame indicator.

TRUE	Frame rectangle provided
FALSE	Client-area rectangle provided.

WinCalcFrameRect Return Value - rc

rc ([BOOL](#)) - returns
Rectangle-calculated indicator.

TRUE	Rectangle successfully calculated
FALSE	Error occurred, or the calculated rectangle is empty.

WinCalcFrameRect - Parameters

hwndFrame ([HWND](#)) - input
Frame-window handle.

prcl ([PRECTL](#)) - in/out
Window rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

fClient ([BOOL](#)) - input
Frame indicator.

TRUE	Frame rectangle provided
FALSE	Client-area rectangle provided.

rc ([BOOL](#)) - returns
Rectangle-calculated indicator.

TRUE	Rectangle successfully calculated
FALSE	Error occurred, or the calculated rectangle is empty.

WinCalcFrameRect - Remarks

In order for WinCalcFrameRect to perform its calculations correctly, the window rectangle in *prcl* must be specified in screen coordinates. The WinMapWindowPoints function is called to make the conversion from client coordinates to screen coordinates, as shown in the example code.

WinCalcFrameRect provides the size and position of the client area within the specified frame rectangle for the specified frame window, or conversely, the size and position of the frame window that would contain a client window of the specified size and position.

WinCalcFrameRect sends a [WM_CALCFRAMERECT](#) message to the frame window. This enables a subclassed frame control to implement the calculation correctly.

This function works if *hwndFrame* is hidden; *hwndFrame* should be hidden if it is required that the window shows a particular client rectangle when the window is first shown.

WinCalcFrameRect - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinCalcFrameRect - Related Functions

Related Functions

- [WinCalcFrameRect](#)
 - [WinCreateFrameControls](#)
 - [WinCreateStdWindow](#)
 - [WinCreateWindow](#)
 - [WinDefWindowProc](#)
 - [WinDestroyWindow](#)
 - [WinQueryClassInfo](#)
 - [WinQueryClassName](#)
 - [WinRegisterClass](#)
 - [WinSubclassWindow](#)
-

WinCalcFrameRect - Related Messages

Related Messages

- [WM_CALCFRAMERECT](#)
-

WinCalcFrameRect - Example Code

This example converts a client window's boundaries into screen coordinates and calls WinCalcFrameRect to calculate an equivalent frame rectangle size.

```
#define INCL_WINFRAMEMGR      /* Window Frame Functions */
#define INCL_WINWINDWMGR     /* Window Manager Functions */
#include <os2.h>

BOOL fSuccess;               /* Success indicator */
HWND hwndClient;             /* client window */
HWND hwndFrame;              /* frame window */
RECTL rclBoundary;           /* Boundary rectangle */

/* convert client boundary coordinates to screen coordinates */
WinMapWindowPoints(hwndClient, HWND_DESKTOP, (PPOINTL)&rclBoundary,
2);

/* calculate equivalent frame boundary from boundary data */
fSuccess = WinCalcFrameRect(hwndFrame, &rclBoundary, FALSE);
```

WinCalcFrameRect - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Related Messages](#)

[Glossary](#)

WinCallMsgFilter

WinCallMsgFilter - Syntax

This function calls a message-filter hook.

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
PQMSG    pqmsg;        /* Message to be passed to the message-filter hook. */
ULONG    ulFilter;     /* Filter. */
BOOL     rc;           /* Message-filter hook return indicator. */

rc = WinCallMsgFilter(hab, pqmsg, ulFilter);
```

WinCallMsgFilter Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinCallMsgFilter Parameter - pqmsg

pqmsg ([PQMSG](#)) - input
Message to be passed to the message-filter hook.

WinCallMsgFilter Parameter - ulFilter

ulFilter (**ULONG**) - input
Filter.

Message-filter code passed to the message-filter hook. This can be an application-specific value or one of the following standard MSGF_* values:

MSGF_DIALOGBOX
Dialog-box mode loop.

MSGF_TRACK
Window-movement and size tracking. When this hook is used the **TRACKINFO** structure specified the *ptiTrackinfo* parameter of the **WinTrackRect** function is updated to give the current state before the hook is called. Only the *rcTrack* and the *fs* parameters are updated.

MSGF_DRAG
Direct manipulation mode loop.

MSGF_DDEPOSTMSG
DDE post message mode loop.

WinCallMsgFilter Return Value - rc

rc (**BOOL**) - returns
Message-filter hook return indicator.

TRUE
A message-filter hook returns TRUE

FALSE
All message-filter hooks return FALSE, or no message-filter hooks are defined.

WinCallMsgFilter - Parameters

hab (**HAB**) - input
Anchor-block handle.

pqmsg (**PQMSG**) - input
Message to be passed to the message-filter hook.

ulFilter (**ULONG**) - input
Filter.

Message-filter code passed to the message-filter hook. This can be an application-specific value or one of the following standard MSGF_* values:

MSGF_DIALOGBOX
Dialog-box mode loop.

MSGF_TRACK
Window-movement and size tracking. When this hook is used the **TRACKINFO** structure specified the *ptiTrackinfo* parameter of the **WinTrackRect** function is updated to give the current state before the hook is called. Only the *rcTrack* and the *fs* parameters are updated.

MSGF_DRAG
Direct manipulation mode loop.

MSGF_DDEPOSTMSG
DDE post message mode loop.

rc ([BOOL](#)) - returns
Message-filter hook return indicator.

TRUE
A message-filter hook returns TRUE

FALSE
All message-filter hooks return FALSE, or no message-filter hooks are defined.

WinCallMsgFilter - Remarks

This function allows an application to pass a message to the message-filter hook procedure(s).

WinCallMsgFilter - Related Functions

Related Functions

- [WinCallMsgFilter](#)
- [WinReleaseHook](#)
- [WinSetHook](#)

WinCallMsgFilter - Example Code

This example calls a message filter hook and passes a WM_CLOSE message in message box mode.

```
#define INCL_WINHOOKS          /* Window Hook Functions          */
#include <os2.h>

BOOL fHookRet;                /* filter hook return indicator */
HAB hab;                      /* Anchor-block handle          */
QMSG pqmsg;                   /* Message to be passed to the */
                               /* message-filter hook           */
ULONG ulFilter;               /* Filter                        */
POINTL ptrPos = {5L,5L};     /* pointer position              */

/* initialize message structure */
pqmsg.hwnd = HWND_DESKTOP;
pqmsg.msg = WM_CLOSE;
pqmsg.mp1 = MPVOID;
pqmsg.mp2 = MPVOID;
pqmsg.time = 0L;
pqmsg.ptl = ptrPos;

/* call hook in message box mode */
ulFilter = MSGF_MESSAGEBOX;

fHookRet = WinCallMsgFilter(hab, &pqmsg, ulFilter);
```

WinCallMsgFilter - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinCancelShutdown

WinCancelShutdown - Syntax

This function cancels a request for an application to shut down.

```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HMQ      hmq;          /* Handle of message queue for current thread. */
BOOL      fCancelAlways; /* Cancellation control. */
BOOL      rc;           /* Success indicator. */

rc = WinCancelShutdown(hmq, fCancelAlways);
```

WinCancelShutdown Parameter - hmq

hmq ([HMQ](#)) - input

Handle of message queue for current thread.

WinCancelShutdown Parameter - fCancelAlways

fCancelAlways ([BOOL](#)) - input

Cancellation control.

TRUE

No [WM_QUIT](#) message should be placed on this queue during system shutdown.

FALSE

The applications ignore any outstanding [WM_QUIT](#) messages already sent to it, but a message should be sent during other system shutdowns.

WinCancelShutdown Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinCancelShutdown - Parameters

hmq (**HMQ**) - input
Handle of message queue for current thread.

fCancelAlways (**BOOL**) - input
Cancellation control.

TRUE	No WM_QUIT message should be placed on this queue during system shutdown.
FALSE	The applications ignore any outstanding WM_QUIT messages already sent to it, but a message should be sent during other system shutdowns.

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinCancelShutdown - Remarks

On a system shutdown, each message queue normally posts a **WM_QUIT** message. An application can then handle this message in one of the three ways:

- Destroy its message queue using `WinDestroyMsgQueue (hmq)`.
- Call `WinCancelShutdown (hmq, TRUE)` to prevent the current thread's event queue from receiving the **WM_QUIT** messages when the system is shut down. This does not stop other threads from receiving this message.
- Call `WinCancelShutdown (hmq, FALSE)` to remove all pending **WM_QUIT** messages from the current thread's event queue but allow future **WM_QUIT** messages to be posted.

Either way, the system can proceed to the next queue.

WinCancelShutdown - Related Functions

Related Functions

- [WinCancelShutdown](#)
- [WinCreateMsgQueue](#)
- [WinInitialize](#)
- [WinTerminate](#)

WinCancelShutdown - Related Messages

Related Messages

- [WM_QUIT](#)

WinCancelShutdown - Example Code

This example cancels a shutdown request (WM_QUIT message) and specifies that the message queue will not accept any new WM_QUIT messages.

```
#define INCL_WINMESSAGEGR  /* Window Message Functions */
#include <os2.h>

HAB  hab,
BOOL fSuccess;           /* Success indicator      */
HMQ  hmq;                /* Queue handle        */

hmq      = WinCreateMsgQueue(hab, 0);
fSuccess = WinCancelShutdown(hmq, TRUE);
```

WinCancelShutdown - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinChangeSwitchEntry

WinChangeSwitchEntry - Syntax

This function changes the information in a Window List entry.

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

H SWITCH    hswitchSwitch; /* Handle to the Window List entry to be changed. */
SWC NTRL    swctlSwitchData; /* Switch-control data. */
ULONG       ulRetCode; /* Return code. */

ulRetCode = WinChangeSwitchEntry(hswitchSwitch,
                                swctlSwitchData);
```

WinChangeSwitchEntry Parameter - hswitchSwitch

hswitchSwitch ([H SWITCH](#)) - input
Handle to the Window List entry to be changed.

WinChangeSwitchEntry Parameter - swctlSwitchData

swctlSwitchData ([SWC NTRL](#)) - input
Switch-control data.

Contains the information to change the Window List entry.

If the *idProcess* field of the [SWC NTRL](#) structure is 0, the current process ID is used.

If the *idSession* field of the [SWC NTRL](#) structure is 0, the current session ID is used.

WinChangeSwitchEntry Return Value - ulRetCode

ulRetCode ([ULONG](#)) - returns
Return code.

0	Successful completion
Other	Error occurred.

WinChangeSwitchEntry - Parameters

hswitchSwitch ([HSWITCH](#)) - input
Handle to the Window List entry to be changed.

swctlSwitchData ([SWCNTRL](#)) - input
Switch-control data.

Contains the information to change the Window List entry.

If the *idProcess* field of the [SWCNTRL](#) structure is 0, the current process ID is used.

If the *idSession* field of the [SWCNTRL](#) structure is 0, the current session ID is used.

ulRetCode ([ULONG](#)) - returns
Return code.

0	Successful completion
Other	Error occurred.

WinChangeSwitchEntry - Remarks

Leading and trailing blanks are removed from the title and, if necessary, it is truncated to 60 characters.

WinChangeSwitchEntry - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_SWITCH_HANDLE (0x1202)
An invalid Window List entry handle was specified.

PMERR_INVALID_WINDOW (0x1206)
The window specified with a Window List call is not a valid frame window.

WinChangeSwitchEntry - Example Code

This example changes the program name of a task-list entry to "Generic: NEW.APP" using the handle returned by WinAddSwitchEntry.

```
#define INCL_WINSWITCHLIST      /* Window Switch List Functions */
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#include <os2.h>

HSWITCH hswitch;               /* task-list entry handle */
SWCNTRL swctl;                 /* switch-control data */
PID pid;                       /* process id */
HAB hab;                       /* anchor-block handle */
HWND hwndFrame;                /* frame handle */

hab = WinQueryAnchorBlock(hwndFrame); /* gets anchor block */
WinQueryWindowProcess(hwndFrame, &pid, NULL); /* gets process id */

/* initialize switch structure */
swctl.hwnd = hwndFrame;         /* window handle */
```

```

swctl.hwndIcon = NULLHANDLE;          /* icon handle          */
swctl.hprog = NULLHANDLE;             /* program handle      */
swctl.idProcess = pid;                /* process identifier  */
swctl.idSession = 0;                 /* session identifier  */
swctl.uchVisibility = SWL_VISIBLE;    /* visibility          */
swctl.fbJump = SWL_JUMPABLE;         /* jump indicator      */
swctl.szSwttitle[0]= 0;              /* program name        */

hswitch = WinCreateSwitchEntry(hab, &swctl);

/* set application name */
strcpy(swctl.szSwttitle, "Generic: NEW.APP");

WinChangeSwitchEntry(hswitch, &swctl);

```

WinChangeSwitchEntry - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

WinCheckButton

WinCheckButton - Syntax

This macro sets the checked state of the specified button control. It returns the previous check state.

```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndDlg;      /* Dialog window handle. */
USHORT    usId;         /* Button control identity. */
USHORT    usChkstate;   /* Indicates the current checked state of the button. */
USHORT    usRetChkst;   /* Returns the previous checkstate. */

usRetChkst = WinCheckButton(hwndDlg, usId,
                             usChkstate);

```

WinCheckButton Parameter - hwndDlg

hwndDlg ([HWND](#)) - input
Dialog window handle.

WinCheckButton Parameter - usId

usId ([USHORT](#)) - input
Button control identity.

WinCheckButton Parameter - usChkstate

usChkstate ([USHORT](#)) - input
Indicates the current checked state of the button.

WinCheckButton Return Value - usRetChkst

usRetChkst ([USHORT](#)) - returns
Returns the previous checkstate.

WinCheckButton - Parameters

hwndDlg ([HWND](#)) - input
Dialog window handle.

usId ([USHORT](#)) - input
Button control identity.

usChkstate ([USHORT](#)) - input
Indicates the current checked state of the button.

usRetChkst ([USHORT](#)) - returns
Returns the previous checkstate.

WinCheckButton - Remarks

This macro expands to:

```
#define WinCheckButton(hwndDlg, usId, usChkstate)
```

```

((USHORT)WinSendDlgItemMsg(hwndDlg,
                             usId,
                             BM_SETCHECK,
                             MPFROMSHORT(usChkstate),
                             (MPARAM)NULL))

```

This call requires the existence of a message queue.

WinCheckButton - Related Functions

Related Functions

- [WinSendDlgItemMsg](#)

WinCheckButton - Related Messages

Related Messages

- [BM_SETCHECK](#)

WinCheckButton - Example Code

This example responds to a button click (BN_CLICKED, WM_CONTROL message) on a check box by setting the checked state of the button.

```

#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#define INCL_WINBUTTONS       /* Window Button definitions */
#include <os2.h>

USHORT usCheckId;             /* check box id */
HWND hwndDlg;                 /* dialog window handle */
USHORT usChkstate;            /* new checked state */
USHORT usOldstate;            /* old checked state */
MPARAM mp1;                   /* Parameter 1 (rectl structure) */
MPARAM mp2;                   /* Parameter 2 (frame boolean) */

case WM_CONTROL:
    /* switch on control code */
    switch(SHORT2FROMMP(mp1))
    {
        case BN_CLICKED:
            usCheckId = SHORT1FROMMP(mp1);

            /* query current check state */
            usChkstate = WinQueryButtonCheckstate(hwndDlg,
                                                    usCheckId);

            /* set box check state */
            usOldstate = WinCheckButton(hwndDlg, usCheckId,
                                         usChkstate);

            break;
    }

```

WinCheckButton - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinCheckInput

WinCheckInput - Syntax

This function is specific to OS/2 Version 2.1 or higher.

This function is used in conjunction with the [MsgInputHook](#) hook to input simulated events into the Presentation Manager input processing system.

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, */  
#include <os2.h>  
  
HAB    hab; /* The application anchor block. */  
BOOL    rc; /* Return code. */  
  
rc = WinCheckInput(hab);
```

WinCheckInput Parameter - hab

hab ([HAB](#)) - input
The application anchor block.

WinCheckInput Return Value - rc

rc ([BOOL](#)) - returns
Return code.

TRUE	The system checked for mouse and keyboard input.
FALSE	An error occurred.

WinCheckInput - Parameters

hab ([HAB](#)) - input
The application anchor block.

rc ([BOOL](#)) - returns
Return code.

TRUE	The system checked for mouse and keyboard input.
FALSE	An error occurred.

WinCheckInput - Remarks

This function must be called whenever an application injects simulated mouse or keyboard events using [MsgInputHook](#), otherwise there is no guarantee when those simulated events will occur. It forces the Presentation Manager input processing system to wake up and take a look at all possible sources of input events: the [MsgInputHook](#) hook, the [JournalPlaybackHook](#) hook, and the Presentation Manager input queue that contains normal mouse and keyboard events.

WinCheckInput - Related Functions

Related Functions

- [MsgInputHook](#)
-

WinCheckInput - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

WinCheckMenuItem

WinCheckMenuItem - Syntax

This macro sets the check state of the specified menu item to the flag.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndMenu; /* Menu window handle. */
USHORT    usId;      /* Item identifier. */
BOOL      fCheck;    /* Check flag. */
BOOL      rc;         /* Success indicator. */

rc = WinCheckMenuItem(hwndMenu, usId, fCheck);
```

WinCheckMenuItem Parameter - hwndMenu

hwndMenu (**HWND**) - input
Menu window handle.

WinCheckMenuItem Parameter - usId

usId (**USHORT**) - input
Item identifier.

WinCheckMenuItem Parameter - fCheck

fCheck (**BOOL**) - input
Check flag.

WinCheckMenuItem Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinCheckMenuItem - Parameters

hwndMenu ([HWND](#)) - input
Menu window handle.

usId ([USHORT](#)) - input
Item identifier.

fCheck ([BOOL](#)) - input
Check flag.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinCheckMenuItem - Remarks

This macro expands to:

```
#define WinCheckMenuItem(hwndMenu, usId, fCheck)
((BOOL)WinSendMsg(hwndMenu,
    MM_SETITEMATTR,
    MPFROM2SHORT(usId, TRUE),
    MPFROM2SHORT(MIA_CHECKED, (BOOL)(fCheck) ? MIA_CHECKED : 0)))
```

This function requires the existence of a message queue.

WinCheckMenuItem - Related Functions

Related Functions

- [WinSendMsg](#)
-

WinCheckMenuItem - Related Messages

Related Messages

- [MM_SETITEMATTR](#)

WinCheckMenuItem - Example Code

This example responds to a select menu message (WM_MENUSELECT) by querying (via WinIsMenuItemChecked) the check attribute and then setting the check state of the menu item that was selected.

```
#define INCL_WINWINDOWMGR          /* Window Manager Functions */
#include <os2.h>

USHORT  usItemId;          /* menu item id */
HWND    hwndMenu;         /* menu handle */
BOOL    usChkstate;       /* new checked state */
BOOL    fSuccess;         /* success indicator */
MPARAM  mp1;              /* Parameter 1 (menu item id) */
MPARAM  mp2;              /* Parameter 2 (menu handle) */

case WM_MENUSELECT:
    usItemId = SHORT1FROMMP(mp1);
    hwndMenu = HWNDFROMMP(mp2);

    /* query current check state */
    usChkstate = WinIsMenuItemChecked(hwndMenu, usItemId);

    /* set menu item check state */
    fSuccess = WinCheckMenuItem(hwndMenu, usItemId, ! usChkstate);
```

WinCheckMenuItem - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinCloseClipbrd

WinCloseClipbrd - Syntax

This function closes the clipboard, allowing other applications to open it with the [WinOpenClipbrd](#) function.

```
#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
```

```
#include <os2.h>

HAB    hab; /* Anchor-block handle. */
BOOL    rc; /* Success indicator. */

rc = WinCloseClipbrd(hab);
```

WinCloseClipbrd Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinCloseClipbrd Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinCloseClipbrd - Parameters

hab (**HAB**) - input
Anchor-block handle.

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinCloseClipbrd - Remarks

This function causes the contents of the clipboard to be drawn in the clipboard viewer window (if any), by sending it a [WM_DRAWCLIPBOARD](#) message. This action occurs only if the clipboard data has changed.

The clipboard must be open before this function is invoked.

WinCloseClipbrd - Related Functions

Related Functions

- [WinCloseClipbrd](#)
- [WinEmptyClipbrd](#)
- [WinEnumClipbrdFmts](#)
- [WinOpenClipbrd](#)
- [WinQueryClipbrdData](#)
- [WinQueryClipbrdFmtInfo](#)
- [WinQueryClipbrdOwner](#)
- [WinQueryClipbrdViewer](#)
- [WinSetClipbrdData](#)
- [WinSetClipbrdOwner](#)
- [WinSetClipbrdViewer](#)

WinCloseClipbrd - Related Messages

Related Messages

- [WM_DRAWCLIPBOARD](#)

WinCloseClipbrd - Example Code

This example closes the clipboard, previously opened by WinOpenClipbrd, to allow other applications to open it for use.

```
#define INCL_WINCLIPBOARD      /* Window Clipboard Functions */
#include <os2.h>

BOOL fSuccess;                /* success indicator */
HAB hab;                      /* anchor-block handle */

fSuccess = WinCloseClipbrd(hab);
```

WinCloseClipbrd - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinCompareStrings

WinCompareStrings - Syntax

Compares two null-terminated strings defined using the same code page.

```
#define INCL_WINCOUNTRY /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;           /* Anchor-block handle. */
ULONG    idCodepage;    /* Code page identity of both strings. */
ULONG    idCountryCode; /* Country code. */
PSZ      pszString1;    /* String 1. */
PSZ      pszString2;    /* String 2. */
ULONG    ulReserved;    /* Reserved value, should be 0. */
ULONG    ulResult;      /* Comparison result. */

ulResult = WinCompareStrings(hab, idCodepage,
                             idCountryCode, pszString1, pszString2,
                             ulReserved);
```

WinCompareStrings Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinCompareStrings Parameter - idCodepage

idCodepage (**ULONG**) - input
Code page identity of both strings.

If *idCodepage* is set to 0, the current code page of the caller is used.

WinCompareStrings Parameter - idCountryCode

idCountryCode (**ULONG**) - input
Country code.

If *idCountryCode* is set to 0, the country information for the default system country code is used.

WinCompareStrings Parameter - pszString1

pszString1 ([PSZ](#)) - input
String 1.

WinCompareStrings Parameter - pszString2

pszString2 ([PSZ](#)) - input
String 2.

WinCompareStrings Parameter - ulReserved

ulReserved ([ULONG](#)) - input
Reserved value, should be 0.

WinCompareStrings Return Value - ulResult

ulResult ([ULONG](#)) - returns
Comparison result.

WCS_EQ	Strings are equal
WCS_LT	String 1 is less than string 2
WCS_GT	String 1 is greater than string 2
WCS_ERROR	Error occurred.

WinCompareStrings - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

idCodepage ([ULONG](#)) - input
Code page identity of both strings.

If *idCodepage* is set to 0, the current code page of the caller is used.

idCountryCode ([ULONG](#)) - input
Country code.

If *idCountryCode* is set to 0, the country information for the default system country code is used.

pszString1 ([PSZ](#)) - input
String 1.

pszString2 ([PSZ](#)) - input
String 2.

ulReserved ([ULONG](#)) - input
Reserved value, should be 0.

ulResult ([ULONG](#)) - returns
Comparison result.

WCS_EQ	Strings are equal
WCS_LT	String 1 is less than string 2
WCS_GT	String 1 is greater than string 2
WCS_ERROR	Error occurred.

WinCompareStrings - Remarks

The first code page of either of the two ASCII code pages specified in CONFIG.SYS is used as the system default. The following is the list of valid code pages:

Country	Code Page
Canadian-French	863
Desktop Publishing	1004
Iceland	861
Latin 1 Multilingual	850
Latin 2 Multilingual	852
Nordic	865
Portuguese	860
Turkey	857
U.S. (IBM PC)	437

Code page 1004 is compatible with Microsoft** Windows**.

The following EBCDIC code pages, based on character set 697, are also available for output:

Country	Code Page
Austrian/German	273
Belgian	500
Brazil	037
Czechoslovakia	870
Danish/Norwegian	277
Finnish/Swedish	278
French	297
Hungary	870
Iceland	871
International	500
Italian	280
Poland	870
Portuguese	037
Spanish	284
Turkey	1026
U.K.-English	285
U.S.-English	037
Yugoslavia	870

Code pages 274 (Belgian) and 282 (Portuguese) can be used to provide access to old data. The following is the list of valid country codes:

Country	Code
Arabic	785
Australian	61
Belgian	32
Canadian-French	2
Danish	45
Finnish	358
French	33
German	49
Hebrew	972
Italian	39
Japanese	81
Korean	82
Latin-American	3
Netherlands	31
Norwegian	47
Portuguese	351
Simpl. Chinese	86
Spanish	34
Swedish	46
Swiss	41
Trad. Chinese	88
UK-English	44
US-English	1
Other country	0

WinCompareStrings - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_STRING_PARM (0x100B)
The specified string parameter is invalid.

WinCompareStrings - Related Functions

Related Functions

- [WinCompareStrings](#)
- [WinLoadString](#)
- [WinNextChar](#)
- [WinPrevChar](#)
- [WinSubstituteStrings](#)
- [WinUpper](#)
- [WinUpperChar](#)

WinCompareStrings - Example Code

This example compares two strings using the same code page: the first string is loaded from a resource DLL, while the second is created by the application.

```
#define INCL_WINCOUNTRY          /* Window Country Functions */
```

```

#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#define INCL_DOSMODULEMGR      /* Module Manager Functions */
#include <os2.h>

ULONG   ulResult;              /* comparison result */
HAB     hab;                   /* anchor-block handle */
ULONG   idCodepage=437; /* Code page identity of both strings */
ULONG   idCountryCode=1; /* Country code */
char     pszString1[10]; /* first string */
char     pszString2[10]; /* second string */
LONG     lLength;              /* length of string */
ULONG   idString = STRING_ID; /* String identifier */
LONG     lBufferMax = 10; /* Size of buffer */
HMODULE  hmodDLL;              /* Handle of the module which contains
                                the help table and help subtable
                                resources. */
CHAR     LoadError[100]; /* object name buffer for DosLoad */
ULONG    rc;                   /* return code */

/* obtain resource handle */
rc = DosLoadModule(LoadError, sizeof(LoadError), "RES.DLL",
                  &hmodDLL);

/* load string from resource */
if (rc == 0)
    lLength = WinLoadString(hab, hmodDLL, idString, lBufferMax,
                          pszString1);

/* compare strings */
if (lLength > 0)
{
    /* set second string */
    strcpy(pszString2, "Compare");

    ulResult = WinCompareStrings(hab, idCodepage, idCountryCode,
                                pszString1, pszString2, 0);
}

```

WinCompareStrings - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinCopyAccelTable

WinCopyAccelTable - Syntax

This function is used to get the accelerator-table data corresponding to an accelerator-table handle, or to determine the size of the

accelerator-table data.

```
#define INCL_WINACCELERATORS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HACCEL      hAccel;          /* Accelerator-table handle. */
PACCELTABLE pacctAccelTable; /* Accelerator-table data area. */
ULONG       ulCopyMax;       /* Maximum data area size. */
ULONG       ulCopied;        /* Amount copied or size required. */

ulCopied = WinCopyAccelTable(hAccel, pacctAccelTable,
                             ulCopyMax);
```

WinCopyAccelTable Parameter - hAccel

hAccel (**HACCEL**) - input
Accelerator-table handle.

WinCopyAccelTable Parameter - pacctAccelTable

pacctAccelTable (**PACCELTABLE**) - in/out
Accelerator-table data area.

- | | |
|-------|--|
| NULL | Return the size, in bytes, of the complete accelerator table, and ignore the <i>ulCopyMax</i> parameter. |
| Other | Copy up to <i>ulCopyMax</i> bytes of the accelerator table into this data area. |

WinCopyAccelTable Parameter - ulCopyMax

ulCopyMax (**ULONG**) - input
Maximum data area size.

WinCopyAccelTable Return Value - ulCopied

ulCopied (**ULONG**) - returns
Amount copied or size required.

- | | |
|-------|---|
| Other | Amount of data copied into the data area, or the size of data area required for the complete accelerator table. |
|-------|---|

0

Error occurred.

WinCopyAccelTable - Parameters

hAccel ([HACCEL](#)) - input
Accelerator-table handle.

pacctAccelTable ([PACCELTABLE](#)) - in/out
Accelerator-table data area.

NULL

Return the size, in bytes, of the complete accelerator table, and ignore the *uiCopyMax* parameter.

Other

Copy up to *uiCopyMax* bytes of the accelerator table into this data area.

uiCopyMax ([ULONG](#)) - input
Maximum data area size.

uiCopied ([ULONG](#)) - returns
Amount copied or size required.

Other

Amount of data copied into the data area, or the size of data area required for the complete accelerator table.

0

Error occurred.

WinCopyAccelTable - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HACCEL (0x101A)
An invalid accelerator-table handle was specified.

WinCopyAccelTable - Related Functions

Related Functions

- [WinCopyAccelTable](#)
- [WinCreateAccelTable](#)
- [WinDestroyAccelTable](#)
- [WinLoadAccelTable](#)
- [WinQueryAccelTable](#)
- [WinSetAccelTable](#)
- [WinTranslateAccel](#)

WinCopyAccelTable - Example Code

This example gets the accelerator-table data corresponding to an accelerator-table handle returned by WinCreateAccelTable or WinLoadAccelTable and assigns the accelerator table code page to a variable.

```
#define INCL_WINACCELERATORS    /* Window Accelerator Functions */
#include <os2.h>

ULONG    ulCopied;           /* bytes copied */
HACCEL   hAccel;            /* Accelerator-table handle */
ACCELTABLE pacctAccelTable; /* Accelerator-table data area */
ULONG    ulCopyMax;         /* Maximum data area size */
ULONG    ulAccelCP;         /* code page */

ulCopyMax = sizeof(pacctAccelTable);
if (hAccel)
    ulCopied = WinCopyAccelTable(hAccel, &pacctAccelTable,
                                ulCopyMax);
if (ulCopied)
    ulAccelCP = pacctAccelTable.codepage;
```

WinCopyAccelTable - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinCopyObject

WinCopyObject - Syntax

This function is specific to OS/2 Version 3.0 or higher.

This function copies an object from its existing location to a specified new destination.

```
#define INCL_WINWORKPLACE
#include <os2.h>

HOBJECT    hObjectofObject; /* Handle of the Workplace Shell object being copied. */
HOBJECT    hObjectofDest;   /* Handle of the destination folder into which hObjectofObject is to be copied. */
ULONG      ulReserved;      /* Reserved value; must be 0. */
HOBJECT    rc;              /* Handle of the newly created object. */

rc = WinCopyObject(hObjectofObject, hObjectofDest,
                   ulReserved);
```

WinCopyObject Parameter - hObjectofObject

hObjectofObject ([HOBJECT](#)) - input
Handle of the Workplace Shell object being copied.

WinCopyObject Parameter - hObjectofDest

hObjectofDest ([HOBJECT](#)) - input
Handle of the destination folder into which *hObjectofObject* is to be copied.

WinCopyObject Parameter - ulReserved

ulReserved ([ULONG](#)) - input
Reserved value; must be 0.

WinCopyObject Return Value - rc

rc ([HOBJECT](#)) - returns
Handle of the newly created object.

A return value of NULLHANDLE indicates that either *hObjectofDest* is NULLHANDLE or an object with the same name as *hObjectofObject* exists in the destination folder.

WinCopyObject - Parameters

hObjectofObject ([HOBJECT](#)) - input
Handle of the Workplace Shell object being copied.

hObjectofDest ([HOBJECT](#)) - input
Handle of the destination folder into which *hObjectofObject* is to be copied.

ulReserved ([ULONG](#)) - input
Reserved value; must be 0.

rc ([HOBJECT](#)) - returns
Handle of the newly created object.

A return value of NULLHANDLE indicates that either *hObjectofDest* is NULLHANDLE or an object with the same name as *hObjectofObject* exists in the destination folder.

WinCopyObject - Remarks

Using [HOBJECT](#)s for .INI files or files in which an application uses a rename/save/delete sequence is not supported. Its REXX counterpart is SysCopyObject.

WinCopyObject - Errors

Possible returns from [WinGetLastError](#)

WPERR_INVALID_FLAGS (0x1719)
An invalid flag was specified.

WinCopyObject - Related Functions

Related Functions

- [WinCreateObject](#)
 - [WinDestroyObject](#)
 - [WinMoveObject](#)
 - [WinQueryObjectWindow](#)
 - [WinSaveObject](#)
-

WinCopyObject - Example Code

This example copies the drives object into the startup folder; the drives folder will be opened at the startup of the system.

```
#define INCL_WINWORKPLACE
#include "os2.h"

HOBJECT    hObjectofDest;
HOBJECT    hObjectofObject;
HOBJECT    hObjectofResult;

hObjectofObject = WinQueryObject("<WP_DRIVES>");
if (hObjectofObject != NULL)
{
    /* WinQueryObject of Drives was successful */
    hObjectofDest = WinQueryObject("<WP_START>");

    if (hObjectofDest != NULL)
    {
        /* WinQueryObject of Startup was successful */
        hObjectofResult = WinCopyObject(hObjectofObject, hObjectofDest, 0);

        if (hObjectofResult != NULL)
        {
            /* Drives Object was successfully copied to the Startup Folder */
```

```
    }
    else
    {
        /* Copy failed */
    }
}
}
```

WinCopyObject - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinCopyRect

WinCopyRect - Syntax

This function copies a rectangle from *prclSrc* to *prclDst*.

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
```

```
HAB      hab;          /* Anchor-block handle. */
PRECTL   prclDst;       /* Destination rectangle. */
PRECTL   prclSrc;       /* Source rectangle. */
BOOL      rc;           /* Success indicator. */
```

```
rc = WinCopyRect(hab, prclDst, prclSrc);
```

WinCopyRect Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinCopyRect Parameter - prclDst

prclDst ([PRECTL](#)) - output
Destination rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

WinCopyRect Parameter - prclSrc

prclSrc ([PRECTL](#)) - input
Source rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

WinCopyRect Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinCopyRect - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

prclDst ([PRECTL](#)) - output
Destination rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

prclSrc ([PRECTL](#)) - input
Source rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinCopyRect - Related Functions

Related Functions

- [WinCopyRect](#)
- [WinEqualRect](#)
- [WinFillRect](#)
- [WinInflateRect](#)
- [WinIntersectRect](#)
- [WinIsRectEmpty](#)
- [WinOffsetRect](#)
- [WinPtInRect](#)
- [WinSetRect](#)
- [WinSetRectEmpty](#)
- [WinSubtractRect](#)
- [WinUnionRect](#)

WinCopyRect - Example Code

This example copies a rectangle using WinCopyRect.

```
#define INCL_WINRECTANGLES      /* Window Rectangle Functions */
#include <os2.h>

BOOL fSuccess;                /* success indicator */
HAB hab;                      /* anchor-block handle */
RECTL prclRect2 = {0,0,200,200}; /* source rectangle */
RECTL prclRect1;              /* destination rectangle */

fSuccess = WinCopyRect(hab, &prclRect1, &prclRect2);
```

WinCopyRect - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinCpTranslateChar

WinCpTranslateChar - Syntax

This function translates a character from one code page to another.

```
#define INCL_WINCOUNTRY /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
ULONG    idCpSource;   /* Source-character code page. */
UCHAR    ucSource;     /* Character to be translated. */
ULONG    idCpDest;     /* Code page of the resultant character. */
UCHAR    ucDest;       /* If nonzero, the translated or substitution (0xFF) character. */

ucDest = WinCpTranslateChar(hab, idCpSource,
                           ucSource, idCpDest);
```

WinCpTranslateChar Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinCpTranslateChar Parameter - idCpSource

idCpSource (**ULONG**) - input
Source-character code page.

WinCpTranslateChar Parameter - ucSource

ucSource (**UCHAR**) - input
Character to be translated.

WinCpTranslateChar Parameter - idCpDest

idCpDest ([ULONG](#)) - input
Code page of the resultant character.

WinCpTranslateChar Return Value - ucDest

ucDest ([UCHAR](#)) - returns
If nonzero, the translated or substitution (0xFF) character.

WinCpTranslateChar - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

idCpSource ([ULONG](#)) - input
Source-character code page.

ucSource ([UCHAR](#)) - input
Character to be translated.

idCpDest ([ULONG](#)) - input
Code page of the resultant character.

ucDest ([UCHAR](#)) - returns
If nonzero, the translated or substitution (0xFF) character.

WinCpTranslateChar - Remarks

Successful invocation of this function indicates that either (1) the character was directly mapped into the destination code page or, (2) the substitution character (0xFF) was returned.

The following is the list of valid code pages:

Country	Code Page
Canadian-French	863
Desktop Publishing	1004
Iceland	861
Latin 1 Multilingual	850
Latin 2 Multilingual	852
Nordic	865
Portuguese	860
Turkey	857
U.S. (IBM PC)	437

Code page 1004 is compatible with Microsoft** Windows**.

The following EBCDIC code pages, based on character set 697, are also available for output:

Country	Code Page
Austrian/German	273
Belgian	500
Brazil	037
Czechoslovakia	870

Danish/Norwegian	277
Finnish/Swedish	278
French	297
Hungary	870
Iceland	871
International	500
Italian	280
Poland	870
Portuguese	037
Spanish	284
Turkey	1026
U.K.-English	285
U.S.-English	037
Yugoslavia	870

Code pages 274 (Belgian) and 282 (Portuguese) can be used to provide access to old data.

WinCpTranslateChar - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_STRING_PARM (0x100B)
The specified string parameter is invalid.

PMERR_INVALID_SRC_CODEPAGE (0x101D)
The source code page parameter is invalid.

PMERR_INVALID_DST_CODEPAGE (0x101E)
The destination code page parameter is invalid.

WinCpTranslateChar - Related Functions

Related Functions

- [WinCpTranslateChar](#)
- [WinCpTranslateString](#)
- [WinQueryCp](#)
- [WinQueryCpList](#)
- [WinSetCp](#)

WinCpTranslateChar - Example Code

This example translates a character from US code page 437 to multilingual code page 850.

```
#define INCL_WINCOUNTRY          /* Window Country Functions */
#include <os2.h>

HAB      hab;                  /* anchor-block handle */
UCHAR    ucDest;               /* translated character */
UCHAR    ucSource = 'A';       /* source character */
ULONG    idCpSource=437;       /* Code page of source character (US) */
ULONG    idCpDest=850;         /* Code page of dest. character (multi) */

ucDest = WinCpTranslateChar(hab, idCpSource, ucSource, idCpDest);
```

WinCpTranslateChar - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinCpTranslateString

WinCpTranslateString - Syntax

This function translates a string from one code page to another.

```
#define INCL_WINCOUNTRY /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
ULONG    idCpSource;   /* Source-string code page. */
PSZ      pszSource;    /* String to be translated. */
ULONG    idCpDest;     /* Code page of the resultant string. */
ULONG    cbLenDest;    /* Maximum length of output string. */
PSZ      pszDest;      /* The translated string. */
BOOL     rc;           /* Success indicator. */

rc = WinCpTranslateString(hab, idCpSource,
    pszSource, idCpDest, cbLenDest, pszDest);
```

WinCpTranslateString Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinCpTranslateString Parameter - idCpSource

idCpSource (ULONG) - input
Source-string code page.

WinCpTranslateString Parameter - pszSource

pszSource (PSZ) - input
String to be translated.

This is a null-terminated string.

WinCpTranslateString Parameter - idCpDest

idCpDest (ULONG) - input
Code page of the resultant string.

WinCpTranslateString Parameter - cbLenDest

cbLenDest (ULONG) - input
Maximum length of output string.

An error is raised if this is not large enough to contain the translated string.

WinCpTranslateString Parameter - pszDest

pszDest (PSZ) - output
The translated string.

This is a null-terminated string.

WinCpTranslateString Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE
Successful completion

FALSE

Error occurred.

WinCpTranslateString - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

idCpSource ([ULONG](#)) - input
Source-string code page.

pszSource ([PSZ](#)) - input
String to be translated.

This is a null-terminated string.

idCpDest ([ULONG](#)) - input
Code page of the resultant string.

cbLenDest ([ULONG](#)) - input
Maximum length of output string.

An error is raised if this is not large enough to contain the translated string.

pszDest ([PSZ](#)) - output
The translated string.

This is a null-terminated string.

rc ([BOOL](#)) - returns
Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinCpTranslateString - Remarks

Successful invocation of this function indicates that either (1) the character was directly mapped into the destination code page or, (2) the substitution character (0xFF) was returned.

The following is the list of valid code pages:

Country	Code Page
Canadian-French	863
Desktop Publishing	1004
Iceland	861
Latin 1 Multilingual	850
Latin 2 Multilingual	852
Nordic	865
Portuguese	860
Turkey	857
U.S. (IBM PC)	437

Code page 1004 is compatible with Microsoft** Windows**.

The following EBCDIC code pages, based on character set 697, are also available for output:

Country	Code Page
Austrian/German	273
Belgian	500
Brazil	037
Czechoslovakia	870
Danish/Norwegian	277
Finnish/Swedish	278
French	297
Hungary	870
Iceland	871
International	500
Italian	280
Poland	870
Portuguese	037
Spanish	284
Turkey	1026
U.K.-English	285
U.S.-English	037
Yugoslavia	870

Code pages 274 (Belgian) and 282 (Portuguese) can be used to provide access to old data.

WinCpTranslateString - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_STRING_PARM (0x100B)
The specified string parameter is invalid.

PMERR_INVALID_SRC_CODEPAGE (0x101D)
The source code page parameter is invalid.

PMERR_INVALID_DST_CODEPAGE (0x101E)
The destination code page parameter is invalid.

PMERR_PARAMETER_OUT_OF_RANGE (0x1003)
The value of a parameter was not within the defined valid range for that parameter.

WinCpTranslateString - Related Functions

Related Functions

- [WinCpTranslateChar](#)
- [WinCpTranslateString](#)
- [WinQueryCp](#)
- [WinQueryCpList](#)
- [WinSetCp](#)

WinCpTranslateString - Example Code

This example translates a string from US code page 437 to multilingual code page 850.

```
#define INCL_WINCOUNTRY          /* Window Country Functions */
#include <os2.h>
```

```

HAB      hab;          /* anchor-block handle          */
ULONG    idCpSource=437; /* Code page of source character (US) */
ULONG    idCpDest=850;  /* Code page of dest. character (multi) */
char      pszSource[10]; /* source string */
char      pszDest[10];  /* destination string */
ULONG     cbLenDest = 9L; /* max length of destination string */
BOOL      fSuccess;
strcpy(pszSource,"TRANSLATE");
fSuccess = WinCpTranslateString(hab, idCpSource, pszSource,
                                idCpDest, cbLenDest, pszDest);

```

WinCpTranslateString - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinCreateAccelTable

WinCreateAccelTable - Syntax

This function creates an accelerator table from the accelerator definitions in memory.

```

#define INCL_WINACCELERATORS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
PACCELTABLE pacctAccelTable; /* Accelerator table. */
HACCEL    haccelhAccel; /* Accelerator-table handle. */

haccelhAccel = WinCreateAccelTable(hab, pacctAccelTable);

```

WinCreateAccelTable Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinCreateAccelTable Parameter - pacctAccelTable

pacctAccelTable ([PACCELTABLE](#)) - input
Accelerator table.

WinCreateAccelTable Return Value - haccelhAccel

haccelhAccel ([HACCEL](#)) - returns
Accelerator-table handle.

WinCreateAccelTable - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

pacctAccelTable ([PACCELTABLE](#)) - input
Accelerator table.

haccelhAccel ([HACCEL](#)) - returns
Accelerator-table handle.

WinCreateAccelTable - Remarks

This function returns a different *haccelhAccel* value when called twice in succession with the same parameter values.

WinCreateAccelTable - Related Functions

Related Functions

- [WinCopyAccelTable](#)
 - [WinCreateAccelTable](#)
 - [WinDestroyAccelTable](#)
 - [WinLoadAccelTable](#)
 - [WinQueryAccelTable](#)
 - [WinSetAccelTable](#)
 - [WinTranslateAccel](#)
-

WinCreateAccelTable - Example Code

This example creates an accelerator-table handle for an in memory accelerator table consisting of 3 accelerator keys, using US codepage 437.

```
#define INCL_WINACCELERATORS    /* Window Accelerator Functions */
#define INCL_WININPUT          /* Key constants */
#define INCL_WINFRAMEMGR      /* Frame control constants */
#include <os2.h>

ULONG      ulAccelLen      = 0;          /* Accelerator-table length */
HACCEL     hAccel         = NULLHANDLE; /* Accelerator-table handle */
PACCELTABLE pacctAccelTable = NULL;      /* Pointer to Accelerator-table */
HAB        hab            = NULLHANDLE; /* Anchor block handle */

ACCEL      acctable[] = {
    AF_SYSCOMMAND | AF_ALT | AF_VIRTUALKEY,VK_F7,SC_MOVE,
    AF_SYSCOMMAND | AF_ALT | AF_VIRTUALKEY,VK_F8,SC_SIZE,
    AF_SYSCOMMAND | AF_ALT | AF_VIRTUALKEY,VK_F12,SC_CLOSE};

    /* Get memory for the Accelerator-table and initialize it. */

    ulAccelLen = sizeof( acctable ) + sizeof( ACCELTABLE );
    pacctAccelTable = (PACCELTABLE) malloc ( ulAccelLen );

    pacctAccelTable->cAccel = 3;          /* Number of ACCEL entries */
    pacctAccelTable->codepage = 437;      /* Code page */
    pacctAccelTable->aaccel[0] = acctable[0]; /* ACCEL entries... */
    pacctAccelTable->aaccel[1] = acctable[1];
    pacctAccelTable->aaccel[2] = acctable[2];

    hAccel = WinCreateAccelTable( hab, pacctAccelTable );
```

WinCreateAccelTable - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinCreateAtomTable

WinCreateAtomTable - Syntax

This function creates a private empty atom table.

```

#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

ULONG      ulInitial;          /* Initial bytes. */
ULONG      ulBuckets;          /* Size of the hash table. */
HATOMTBL   hatomtblAtomTbl;    /* Atom-table handle. */

hatomtblAtomTbl = WinCreateAtomTable(ulInitial,
                                     ulBuckets);

```

WinCreateAtomTable Parameter - ulInitial

ulInitial (**ULONG**) - input
Initial bytes.

This parameter is ignored for **version 3, or higher**, of the OS/2 operating system. The *ulInitial* parameter is dynamically allocated. The initial number of bytes should be set to zero.

For **versions prior to version 3** of the OS/2 operating system, this parameter is the initial number of bytes to be reserved for the atom table. This is a lower bound on the amount of memory reserved. The amount of memory actually used by an atom table depends upon the actual number of atoms stored in the table. If zero, the size of the atom table is the minimum size needed to store the atom hash table. The minimum size of atom table allocated is $16 + (2 * ulBuckets)$.

WinCreateAtomTable Parameter - ulBuckets

ulBuckets (**ULONG**) - input
Size of the hash table.

Used to access atoms. If zero, the default value of 37 is used. The best results are achieved if a prime number is used.

WinCreateAtomTable Return Value - hatomtblAtomTbl

hatomtblAtomTbl (**HATOMTBL**) - returns
Atom-table handle.

NULLHANDLE

Call failed.

Other

Atom-table handle. This must be passed as a parameter in subsequent atom manager calls.

WinCreateAtomTable - Parameters

ulInitial (**ULONG**) - input

Initial bytes.

This parameter is ignored for **version 3, or higher**, of the OS/2 operating system. The *ulInitial* parameter is dynamically allocated. The initial number of bytes should be set to zero.

For **versions prior to version 3** of the OS/2 operating system, this parameter is the initial number of bytes to be reserved for the atom table. This is a lower bound on the amount of memory reserved. The amount of memory actually used by an atom table depends upon the actual number of atoms stored in the table. If zero, the size of the atom table is the minimum size needed to store the atom hash table. The minimum size of atom table allocated is $16+(2*ulBuckets)$.

ulBuckets (ULONG) - input
Size of the hash table.

Used to access atoms. If zero, the default value of 37 is used. The best results are achieved if a prime number is used.

hatomtblAtomTbl (HATOMTBL) - returns
Atom-table handle.

NULLHANDLE

Call failed.

Other

Atom-table handle. This must be passed as a parameter in subsequent atom manager calls.

WinCreateAtomTable - Remarks

The atom table is owned by the process from which this function is issued. It cannot be accessed directly from any other process. If it is still in existence when the process terminates, it will automatically be deleted by the system.

There is a system atom table which is created at boot time, which cannot be destroyed, and which can be accessed by any process in the system. The handle of the system atom table is queried with the [WinQuerySystemAtomTable](#) function.

WinCreateAtomTable - Related Functions

Related Functions

- [WinAddAtom](#)
- [WinCreateAtomTable](#)
- [WinDeleteAtom](#)
- [WinDestroyAtomTable](#)
- [WinFindAtom](#)
- [WinQueryAtomLength](#)
- [WinQueryAtomName](#)
- [WinQueryAtomUsage](#)
- [WinQuerySystemAtomTable](#)

WinCreateAtomTable - Example Code

This example creates an Atom Table of default size, adds the atom "newatom" to the new table, and then destroys the table.

```
#define INCL_WINATOM          /* Window Atom functions          */
#include <os2.h>

ATOM      atom;              /* New atom value              */
HATOMTBL  hatomtblAtomTbl;   /* Atom-table handle           */
HATOMTBL  hatomtblDestroy;   /* Result of destroy table     */
```

```

char      pszAtomName[10];    /* Atom name */
ULONG     ulInitial = 0;      /* Initial atom table size (use default) */
ULONG     ulBuckets = 0;      /* Size of hash table (use default) */

/* Create an atom table of the default size */
hAtomTblAtomTbl = WinCreateAtomTable(ulInitial, ulBuckets);

/* Define the name of the new atom and add it to the table */
strcpy(pszAtomName, "newatom");
atom = WinAddAtom(hAtomTblAtomTbl, pszAtomName);
hAtomTblDestroy = WinDestroyAtomTable(hAtomTblAtomTbl);

```

WinCreateAtomTable - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinCreateCursor

WinCreateCursor - Syntax

This function creates or changes a cursor for a specified window.

```

#define INCL_WINCursors /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND     hwnd;    /* Handle of window in which cursor is displayed. */
LONG     lx;      /* x-position of cursor. */
LONG     ly;      /* y-position of cursor. */
LONG     lcx;     /* x-size of cursor. */
LONG     lcy;     /* y-size of cursor. */
ULONG     ulrgf;   /* Controls the appearance of the cursor. */
PRECTL   prclClip; /* Cursor rectangle. */
BOOL     rc;       /* Success indicator. */

rc = WinCreateCursor(hwnd, lx, ly, lcx, lcy,
    ulrgf, prclClip);

```

WinCreateCursor Parameter - hwnd

hwnd (**HWND**) - input

Handle of window in which cursor is displayed.

This must be the handle of a window for which the application can receive input.

WinCreateCursor Parameter - lx

lx (**LONG**) - input

x-position of cursor.

WinCreateCursor Parameter - ly

ly (**LONG**) - input

y-position of cursor.

WinCreateCursor Parameter - lcx

lcx (**LONG**) - input

x-size of cursor.

If 0, the system nominal border width (SV_CXBORDER) is used.

WinCreateCursor Parameter - lcy

lcy (**LONG**) - input

y-size of cursor.

If 0, the system nominal border height (SV_CYBORDER) is used.

WinCreateCursor Parameter - ulrgf

ulrgf (**ULONG**) - input

Controls the appearance of the cursor.

CURSOR_SOLID

The cursor is solid.

CURSOR_HALFTONE

The cursor is halftone.

CURSOR_FRAME

The cursor is a rectangular frame.

CURSOR_FLASH

The cursor flashes.

CURSOR_SETPOS

Set a new cursor position. *lx* and *ly* are ignored. Used when a cursor has already been created. In this case, all other appearance flags are ignored.

WinCreateCursor Parameter - prclClip

prclClip ([PRECTL](#)) - input

Cursor rectangle.

A rectangle within which the cursor is visible. If the cursor goes outside this rectangle, it is clipped away and is invisible.

The rectangle is specified in window coordinates.

If *prclClip* is NULL, the drawing of the cursor is clipped to the window rectangle of *hwnd*.

Note: The cursor is always clipped to the window rectangle, even if part of *prclClip* is outside it.

The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

WinCreateCursor Return Value - rc

rc ([BOOL](#)) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinCreateCursor - Parameters

hwnd ([HWND](#)) - input

Handle of window in which cursor is displayed.

This must be the handle of a window for which the application can receive input.

lx ([LONG](#)) - input

x-position of cursor.

ly ([LONG](#)) - input
y-position of cursor.

lcx ([LONG](#)) - input
x-size of cursor.

If 0, the system nominal border width (SV_CXBORDER) is used.

lcy ([LONG](#)) - input
y-size of cursor.

If 0, the system nominal border height (SV_CYBORDER) is used.

ulrgf ([ULONG](#)) - input
Controls the appearance of the cursor.

CURSOR_SOLID
The cursor is solid.

CURSOR_HALFTONE
The cursor is halftone.

CURSOR_FRAME
The cursor is a rectangular frame.

CURSOR_FLASH
The cursor flashes.

CURSOR_SETPOS
Set a new cursor position. *lcx* and *lcy* are ignored. Used when a cursor has already been created. In this case, all other appearance flags are ignored.

prclClip ([PRECTL](#)) - input
Cursor rectangle.

A rectangle within which the cursor is visible. If the cursor goes outside this rectangle, it is clipped away and is invisible.

The rectangle is specified in window coordinates.

If *prclClip* is NULL, the drawing of the cursor is clipped to the window rectangle of *hwnd*.

Note: The cursor is always clipped to the window rectangle, even if part of *prclClip* is outside it.

The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

rc ([BOOL](#)) - returns
Success indicator.

TRUE
Successful completion

FALSE
Error occurred.

WinCreateCursor - Remarks

The cursor is used to indicate the position of text input. It is initially hidden and must be made visible using the [WinShowCursor](#) function.

This function destroys any existing cursor, as it is confusing to the user if two cursors are visible at any one time. An application creates and displays a cursor when it has the input focus, or is the active window. Creating a cursor at any other time can stop the cursor from flashing in another window. Similarly, when the application loses the input focus or becomes inactive, it destroys its cursor using the [WinDestroyCursor](#) function.

The cursor width is generally specified as 0 (nominal border width is used). This is preferable to a value of 1, for example, as such a fine width is almost invisible on a high-resolution device.

When *ulrgf* is set to CURSOR_FLASH, TID_CURSOR timer is created.

WinCreateCursor - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_INVALID_FLAG (0x1019)
An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

WinCreateCursor - Related Functions

Related Functions

- [WinCreateCursor](#)
 - [WinDestroyCursor](#)
 - [WinQueryCursorInfo](#)
 - [WinShowCursor](#)
-

WinCreateCursor - Example Code

This example creates a cursor of default height and width at (0,200) which will be visible within the entirety of the input window.

```
#define INCL_WINCursors      /* Window Cursor Functions      */
#include <os2.h>

BOOL fSuccess;              /* success indicator      */
HWND hwnd;                  /* cursor display window  */
LONG lx = 0;                /* cursor x position      */
LONG ly = 200;              /* cursor y position      */
ULONG ulrgf;                /* cursor appearance      */

fSuccess = WinCreateCursor(hwnd, lx, ly, 0, 0, ulrgf, NULL);
```

WinCreateCursor - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinCreateDlg

WinCreateDlg - Syntax

This function creates a dialog window.

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndParent;    /* Parent-window handle of the created dialog window. */
HWND      hwndOwner;     /* Requested owner-window handle of the created dialog window. */
PFNWND    pfnDlgProc;    /* Dialog procedure for the created dialog window. */
PDLGTEMPLATE pdlgt;      /* Dialog template. */
PVOID     pCreateParams; /* Pointer to application-defined data area. */
HWND      hwndDlg;       /* Dialog-window handle. */

hwndDlg = WinCreateDlg(hwndParent, hwndOwner,
                      pfnDlgProc, pdlgt, pCreateParams);
```

WinCreateDlg Parameter - hwndParent

hwndParent ([HWND](#)) - input
Parent-window handle of the created dialog window.

HWND_DESKTOP	The desktop window
HWND_OBJECT	Object window
Other	Specified window.

WinCreateDlg Parameter - hwndOwner

hwndOwner ([HWND](#)) - input
Requested owner-window handle of the created dialog window.

The actual owner window is calculated using the algorithm specified in the description of the [WinLoadDlg](#) function.

WinCreateDlg Parameter - pfnDlgProc

pfnDlgProc ([PFNWP](#)) - input
Dialog procedure for the created dialog window.

WinCreateDlg Parameter - pDlg

pDlg ([PDLGTEMPLATE](#)) - input
Dialog template.

WinCreateDlg Parameter - pCreateParams

pCreateParams ([PVOID](#)) - input
Pointer to application-defined data area.

This is passed to the dialog procedure in the [WM_INITDLG](#) message.

This parameter **MUST** be a pointer rather than a long.

WinCreateDlg Return Value - hwndDlg

hwndDlg ([HWND](#)) - returns
Dialog-window handle.

NULLHANDLE	Dialog window not created
Other	Dialog-window handle.

WinCreateDlg - Parameters

hwndParent ([HWND](#)) - input
Parent-window handle of the created dialog window.

HWND_DESKTOP	The desktop window
HWND_OBJECT	Object window
Other	Specified window.

hwndOwner ([HWND](#)) - input
Requested owner-window handle of the created dialog window.

The actual owner window is calculated using the algorithm specified in the description of the [WinLoadDlg](#) function.

pfnDlgProc ([PFNWP](#)) - input
Dialog procedure for the created dialog window.

pdlg ([PDLGTEMPLATE](#)) - input
Dialog template.

pCreateParams ([PVOID](#)) - input
Pointer to application-defined data area.

This is passed to the dialog procedure in the [WM_INITDLG](#) message.

This parameter MUST be a pointer rather than a long.

hwndDlg ([HWND](#)) - returns
Dialog-window handle.

NULLHANDLE	Dialog window not created
Other	Dialog-window handle.

WinCreateDlg - Remarks

This function is identical to the [WinLoadDlg](#) call except that it creates a dialog window from *pdlg* in memory, rather than a dialog template in a resource file.

This function should not be used while the pointing device capture is set (see [WinSetCapture](#)).

WinCreateDlg - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_INVALID_INTEGER_ATOM (0x1016)
The specified atom is not a valid integer atom.

PMERR_INVALID_ATOM_NAME (0x1015)
An invalid atom name string was passed.

PMERR_ATOM_NAME_NOT_FOUND (0x1017)
The specified atom name is not in the atom table.

WinCreateDlg - Related Functions

Related Functions

- [WinCreateDlg](#)
- [WinDefDlgProc](#)
- [WinDismissDlg](#)
- [WinDlgBox](#)

- [WinGetDlgMsg](#)
- [WinLoadDlg](#)
- [WinProcessDlg](#)

WinCreateDlg - Related Messages

Related Messages

- [WM_INITDLG](#)

WinCreateDlg - Example Code

This example loads a dialog template from the application's resources and uses the template with the WinCreateDlg function to create a dialog window. This example is identical to calling the WinLoadDlg function, but gives the application the advantage of reviewing and modifying the dialog template before creating the dialog window.

```
#define INCL_WINDIALOGS          /* Window Dialog Mgr Functions */
#define INCL_DOSRESOURCES        /* OS/2 Resource functions */
#include <os2.h>

PFNWP    MyDlgProc;
#define   ID_DIALOG  1

PDLGTEMPLATE pdlgt;          /* dialog template */

DosGetResource(0L, RT_DIALOG, ID_DIALOG, (PVOID)pdlgt);

/* make any changes to dialog template here */

WinCreateDlg(HWND_DESKTOP,
    NULLHANDLE, /* owner window */
    MyDlgProc,  /* address of dialog procedure */
    pdlgt,      /* address of dialog structure */
    NULL);      /* application-specific data */
```

WinCreateDlg - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinCreateFrameControls

WinCreateFrameControls - Syntax

This function creates the standard frame controls for a specified window.

```
#define INCL_WINFRAMEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndFrame; /* Frame-window handle. */
PFRAMECDATA pfcdata; /* Frame-control data. */
PSZ       pszTitle; /* Title string. */
BOOL      rc; /* Success indicator. */

rc = WinCreateFrameControls(hwndFrame, pfcdata,
                           pszTitle);
```

WinCreateFrameControls Parameter - hwndFrame

hwndFrame (**HWND**) - input
Frame-window handle.

Becomes the parent and owner window of all the frame controls that are created:

HWND_DESKTOP	The desktop window
HWND_OBJECT	Object window
Other	Specified window.

WinCreateFrameControls Parameter - pfcdata

pfcdata (**PFRAMECDATA**) - input
Frame-control data.

This includes a combination of frame creation flags (FCF_*), that specifies which frame controls are to be created. For further information, see

WinCreateFrameControls Parameter - pszTitle

pszTitle (**PSZ**) - input

Title string.

This parameter contains a string that is displayed in the WC_TITLEBAR control only when FCF_TITLEBAR is specified in *pfcdata*.

WinCreateFrameControls Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinCreateFrameControls - Parameters

hwndFrame (**HWND**) - input
Frame-window handle.

Becomes the parent and owner window of all the frame controls that are created:

HWND_DESKTOP	The desktop window
HWND_OBJECT	Object window
Other	Specified window.

pfcdata (**PFRAMECDATA**) - input
Frame-control data.

This includes a combination of frame creation flags (FCF_*), that specifies which frame controls are to be created. For further information, see

pszTitle (**PSZ**) - input
Title string.

This parameter contains a string that is displayed in the WC_TITLEBAR control only when FCF_TITLEBAR is specified in *pfcdata*.

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinCreateFrameControls - Remarks

This function is typically used when the standard frame controls are needed for use with a nonstandard window (such as a non WC_FRAME class).

All of the controls are created with the standard FID_* window identifiers; see

The controls are created but not formatted. Formatting must be done by setting the required positions. All controls are created with position and size set to zero and WS_VISIBLE is not set.

WinCreateFrameControls - Errors

Possible returns from `WinGetLastError`

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_INVALID_FLAG (0x1019)
An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

WinCreateFrameControls - Related Functions

Related Functions

- [WinCalcFrameRect](#)
- [WinCreateFrameControls](#)
- [WinCreateStdWindow](#)
- [WinCreateWindow](#)
- [WinDefWindowProc](#)
- [WinDestroyWindow](#)
- [WinQueryClassInfo](#)
- [WinQueryClassName](#)
- [WinRegisterClass](#)
- [WinSubclassWindow](#)

WinCreateFrameControls - Example Code

This example creates frame controls (title bar, system menu, size border, and min/max buttons) for a button window created by `WinCreateWindow`. The new controls are owned by and children of the button window.

```
#define INCL_WINWINDOWMGR           /* Window Manager Functions */
#define INCL_WINLISTBOXES          /* List Box definitions */
#define INCL_WINFRAMEMGR           /* Frame Manager Functions */
#include <os2.h>

HWND      hwnd;                    /* cursor display window */
ULONG flStyle;                     /* window style */
USHORT ButtonId;                   /* window id (app supplied) */
BOOL fSuccess;                     /* success indicator */
FRAMECDATA pFcdData;              /* Frame-control data */
USHORT usFrameId;                  /* frame resource id (app supplied) */

flStyle = WS_VISIBLE;              /* create window visible */

/* create button window (no frame controls) */
hwnd = WinCreateWindow(HWND_DESKTOP, /* parent window */
                       WC_BUTTON,    /* class name */
                       "new button", /* window text */
                       flStyle,       /* window style */
                       0, 0,          /* position (x,y) */
                       200, 100,     /* size (width,height) */
                       0, 0, 0, 0, 0 /* style, title, icon, cursor, menu, etc. */
                       );
```



```

                                OL,                /* owner window          */
                                HWND_TOP,           /* sibling window         */
                                ButtonId,          /* window id             */
                                NULL,              /* control data          */
                                NULL);             /* presentation parms    */

/*****
 * initialize frame structure *
 *****/
pFcdata.cb = sizeof(FRAMECDATA); /* Length */
/* Frame-creation flags */
pFcdata.flCreateFlags = FCF_TITLEBAR | FCF_SYSMENU |
                        FCF_SIZEBORDER | FCF_MINMAX;
pFcdata.hmodResources = 0L;      /* resource in EXE */
pFcdata.idResources = usFrameId; /* resource id */

/* create frame controls; display 'button frame' on title bar */
fSuccess = WinCreateFrameControls(hwnd, &pFcdata, "button frame");

```

WinCreateFrameControls - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinCreateHelpInstance

WinCreateHelpInstance - Syntax

This function creates an instance of the Help Manager with which to request Help Manager functions.

```

#define INCL_WINHELP /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;
PHELPINIT phinitHMIInitStructure; /* Anchor-block handle. */
HWND     hwndhelp;                /* Help Manager initialization structure. */

hwndhelp = WinCreateHelpInstance(hab, phinitHMIInitStructure);

```

WinCreateHelpInstance Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

The handle of the application anchor block returned from the [WinInitialize](#) function.

WinCreateHelpInstance Parameter - phinitHMInitStructure

phinitHMInitStructure ([PHELPINIT](#)) - in/out
Help Manager initialization structure.

WinCreateHelpInstance Return Value - hwndhelp

hwndhelp ([HWND](#)) - returns
Help Manager handle.

NULLHANDLE	Error occurred
Other	Help Manager handle.

WinCreateHelpInstance - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

The handle of the application anchor block returned from the [WinInitialize](#) function.

phinitHMInitStructure ([PHELPINIT](#)) - in/out
Help Manager initialization structure.

hwndhelp ([HWND](#)) - returns
Help Manager handle.

NULLHANDLE	Error occurred
Other	Help Manager handle.

WinCreateHelpInstance - Remarks

If an error occurs, it is in the parameter of the [HELPINIT](#) structure.

WinCreateHelpInstance - Related Functions

Related Functions

- [WinAssociateHelpInstance](#)
 - [WinCreateHelpInstance](#)
 - [WinCreateHelpTable](#)
 - [WinDestroyHelpInstance](#)
 - [WinLoadHelpTable](#)
 - [WinQueryHelpInstance](#)
-

WinCreateHelpInstance - Example Code

This example shows a typical main function for an application which uses help. Following creation of the main application window the Help Manager is initialized and associated with the window. The help table is defined in the application's resources. When the window is destroyed, terminating the application, the help instance is also destroyed.

```
#define INCL=_WIN
#include <os2.h>

#define IDHT_APPLICATION      100      /* id of HELP TABLE in resource file
*/

main( int argc, char *argv[], char *envp[] )
{
    HAB hab = WinInitialize( 0 );
    HMQ hmq = WinCreateMsgQueue( hab, 0 );
    HWND hwnd;
    HWND hwndClient;
    HWND hwndHelp;
    QMSG qmsg;
    ULONG flStyle;
    HELPINIT helpinit;

    /* Setup the help initialization structure */
    helpinit.cb = sizeof( HELPINIT );
    helpinit.ulReturnCode = 0L;
    helpinit.pszTutorialName = (PSZ)NULL;
    /* Help table in application resource */
    helpinit.phtHelpTable = (PHELPTABLE)MAKEULONG( IDHT_APPLICATION, 0xffff );
    helpinit.hmodHelpTableModule = NULLHANDLE;
    /* Default action bar and accelerators */
    helpinit.hmodAccelActionBarModule = NULLHANDLE;
    helpinit.idAccelTable = 0;
    helpinit.idActionBar = 0;
    helpinit.pszHelpWindowTitle = "APPNAME HELP";
    helpinit.fShowPanelId = CMIC_SHOW_PANEL_ID;
    helpinit.pszHelpLibraryName = "APPNAME.HLP";

    /* Register the class */
    if( WinRegisterClass( ... ) )
    {
        /* create the main window */
        flStyle = FCF_STANDARD;
        hwnd = WinCreateStdWindow( ... );

        if( hwnd )
        {
            /* Create and associate the help instance */
            hwndHelp = WinCreateHelpInstance( hab, &helpinit );

            if( hwndHelp && WinAssociateHelpInstance( hwndHelp, hwnd ) )
            {
                /* Process messages */
                while( WinGetMsg( hab, &qmsg, NULLHANDLE, 0, 0 ) )
```

```

        {
            WinDispatchMsg( hab, &qmsg );
        } /* endwhile */
    }

    /* Remove help instance - note: add */
    /* WinAssociateHelpInstance( NULLHANDLE, hwnd ); */
    /* to WM_DESTROY processing to remove the association. */
    WinDestroyHelpInstance( hwndHelp );
}

/* finish the cleanup and exit */
WinDestroyMsgQueue( hmq );
WinTerminate( hab );
}

```

WinCreateHelpInstance - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinCreateHelpTable

WinCreateHelpTable - Syntax

This function is used to identify or change the help table.

```

#define INCL_WINHELP /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndHelpInstance; /* Handle of an instance of the Help Manager. */
PHELPTEXT phtHelpTable;     /* Help table allocated by the application. */
BOOL      rc;                /* Success indicator. */

rc = WinCreateHelpTable(hwndHelpInstance,
                        phtHelpTable);

```

WinCreateHelpTable Parameter - hwndHelpInstance

hwndHelpInstance ([HWND](#)) - input
Handle of an instance of the Help Manager.

This is the handle returned by the [WinCreateHelpInstance](#) call.

WinCreateHelpTable Parameter - phtHelpTable

phtHelpTable ([PHELP_TABLE](#)) - input
Help table allocated by the application.

WinCreateHelpTable Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinCreateHelpTable - Parameters

hwndHelpInstance ([HWND](#)) - input
Handle of an instance of the Help Manager.

This is the handle returned by the [WinCreateHelpInstance](#) call.

phtHelpTable ([PHELP_TABLE](#)) - input
Help table allocated by the application.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinCreateHelpTable - Remarks

This function corresponds to the [HM_CREATE_HELP_TABLE](#) message that identifies a help table that is in memory.

WinCreateHelpTable - Related Functions

Related Functions

- [WinAssociateHelpInstance](#)
- [WinCreateHelpInstance](#)
- [WinCreateHelpTable](#)
- [WinDestroyHelpInstance](#)
- [WinLoadHelpTable](#)
- [WinQueryHelpInstance](#)

WinCreateHelpTable - Related Messages

Related Messages

- [HM_CREATE_HELP_TABLE](#)

WinCreateHelpTable - Example Code

This example creates a help table in memory and passes the table to the Help Manager via WinCreateHelpTable. The help instance must have been created by WinCreateHelpInstance.

```
#define INCL_WINHELP
#include <os2.h>

/* DEFINES for window id's, menu items, controls, panels, etc. should */
/* be inserted here or in additional include files. */

/* Subtable for the main window's help */
HELPSUBTABLE phtMainTable[] = { 2, /* Length of each entry */
/* Fill in one line for each menu item */
IDM_FILE, PANELID_FILEMENU,
IDM_FILENEW, PANELID_FILENEW,
IDM_FILEOPEN, PANELID_FILEOPEN,
IDM_FILESAVE, PANELID_FILESAVE,
IDM_FILESAVEAS, PANELID_FILESAVEAS,
IDM_FILEEXIT, PANELID_FILEEXIT };

/* Subtable for the dialog window's help */
HELPSUBTABLE phtDlgTable[] = { 2, /* Length of each entry */
/* Fill in one line for each control */
IDC_EDITFLD, PANELID_DLGEDITFLD,
IDC_OK, PANELID_DLGOK,
IDC_CANCEL, PANELID_DLGCANCEL,
IDC_HELP, PANELID_HELP };

/* Help table for the applications context sensitive help */
HELPTABLE phtHelpTable[] = { WINDOWID_MAIN, phtMainTable,
PANELID_MAINEXT,
WINDOWID_DLG, phtDlgTable, PANELID_DLGEEXT,
0, NULL, 0 };

BOOL CreateHelpTable( HWND hWnd )
{
    BOOL bSuccess = FALSE;
    HWND hwndHelp;

    /* Get the associated help instance */
    hwndHelp = WinQueryHelpInstance( hWnd );

    if( hwndHelp )
    {
        /* Pass address of help table to the Help Manager */
    }
}
```

```

        bSuccess = WinCreateHelpTable( hwndHelp, phtHelpTable );
    }

    /* return success indicator */
    return bSuccess;
}

```

WinCreateHelpTable - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinCreateMenu

WinCreateMenu - Syntax

This function creates a menu window from the menu template.

```

#define INCL_WINMENUS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndParent; /* Owner- and parent-window handle of the created menu window. */
PVOID     lpmt;       /* Menu template in binary format. */
HWND      hwndMenu;   /* Menu-window handle. */

hwndMenu = WinCreateMenu(hwndParent, lpmt);

```

WinCreateMenu Parameter - hwndParent

hwndParent ([HWND](#)) - input

Owner- and parent-window handle of the created menu window.

If this is `HWND_OBJECT` or a window handle returned by the [WinQueryObjectWindow](#) call, the menu window is created as an object window.

`HWND_DESKTOP`
The desktop window

HWND_OBJECT Object window
Other Specified window.

WinCreateMenu Parameter - lpmt

lpmt ([PVOID](#)) - input
Menu template in binary format.

WinCreateMenu Return Value - hwndMenu

hwndMenu ([HWND](#)) - returns
Menu-window handle.

WinCreateMenu - Parameters

hwndParent ([HWND](#)) - input
Owner- and parent-window handle of the created menu window.

If this is HWND_OBJECT or a window handle returned by the [WinQueryObjectWindow](#) call, the menu window is created as an object window.

HWND_DESKTOP The desktop window
HWND_OBJECT Object window
Other Specified window.

lpmt ([PVOID](#)) - input
Menu template in binary format.

hwndMenu ([HWND](#)) - returns
Menu-window handle.

WinCreateMenu - Remarks

The menu window is created with an identity of FID_MENU.

WinCreateMenu - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinCreateMenu - Related Functions

Related Functions

- [WinCreateMenu](#)
- [WinLoadMenu](#)
- [WinPopupMenu](#)

WinCreateMenu - Example Code

This code will load a menu template from a dll and then use it to add a menu to the frame window hwndFrame which has been previously created without a menu.

```
HMOD      hmod;
HWND      hwndFrame, hwndMenu;
USHORT    idMenu = 999;
BYTE      lpmt;

DosLoadModule(NULL, 0, "MYDLL.DLL", &hmod);

/* Load menu template */
DosGetResource2(hmod, (USHORT)RT_MENU, idMenu, &lpmt);

hwndMenu = WinCreateMenu(hwndFrame, lpmt); /* Create a menu */

DosFreeResource(lpmt); /* free menu template resource */
```

WinCreateMenu - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinCreateMsgQueue

WinCreateMsgQueue - Syntax

This function creates a message queue.

```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
LONG     lQueueSize;   /* Maximum queue size. */
HMQ      hmq;          /* Message-queue handle. */

hmq = WinCreateMsgQueue(hab, lQueueSize);
```

WinCreateMsgQueue Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinCreateMsgQueue Parameter - lQueueSize

lQueueSize (**LONG**) - input
Maximum queue size.

This parameter is ignored for **versions higher than version 3** of the OS/2 operating system. The *lQueueSize* parameter is dynamically allocated. The maximum queue size should be set to zero.

For **version 3 and lower** of the OS/2 operating system, this parameter is the maximum number of messages that can be queued.

0 Use the system default queue size which is 10 messages.

Other Maximum queue size.

WinCreateMsgQueue Return Value - hmq

hmq (**HMQ**) - returns
Message-queue handle.

NULLHANDLE Queue cannot be created.

Other Message-queue handle.

WinCreateMsgQueue - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

IQueueSize ([LONG](#)) - input
Maximum queue size.

This parameter is ignored for **versions higher than version 3** of the OS/2 operating system. The *IQueueSize* parameter is dynamically allocated. The maximum queue size should be set to zero.

For **version 3 and lower** of the OS/2 operating system, this parameter is the maximum number of messages that can be queued.

0 Use the system default queue size which is 10 messages.

Other Maximum queue size.

hmq ([HMQ](#)) - returns
Message-queue handle.

NULLHANDLE Queue cannot be created.

Other Message-queue handle.

WinCreateMsgQueue - Remarks

Most PM calls require a message queue. WinCreateMsgQueue must be issued after the [WinInitialize](#) function, but before any other PM calls are invoked. It must be issued only once per thread.

WinCreateMsgQueue - Errors

Possible returns from [WinGetLastError](#)

PMERR_HEAP_MAX_SIZE_REACHED (0x1012)
The heap has reached its maximum size (64KB), and cannot be increased.

PMERR_HEAP_OUT_OF_MEMORY (0x1011)
An attempt to increase the size of the heap failed.

PMERR_QUEUE_TOO_LARGE (0x1018)
An attempt to create a message queue has failed because the value specified for the size of the message queue is too large.

PMERR_RESOURCE_NOT_FOUND (0x100A)
The specified resource identity could not be found.

PMERR_NOT_IN_A_PM_SESSION (0x1051)
An attempt was made to access function that is only available from PM programs from a non-PM session.

PMERR_MSG_QUEUE_ALREADY_EXISTS (0x1052)
An attempt to create a message queue for a thread failed because a message queue already exists for the calling thread.

WinCreateMsgQueue - Related Functions

Related Functions

- [WinBroadcastMsg](#)
- [WinCancelShutdown](#)
- [WinDestroyMsgQueue](#)
- [WinDispatchMsg](#)
- [WinGetDlgMsg](#)
- [WinGetMsg](#)
- [WinInitialize](#)
- [WinInSendMessage](#)
- [WinPeekMsg](#)
- [WinPostMsg](#)
- [WinPostQueueMsg](#)
- [WinQueryMsgPos](#)
- [WinQueryMsgTime](#)
- [WinQueryQueueInfo](#)
- [WinQueryQueueStatus](#)
- [WinSendDlgItemMsg](#)
- [WinSendMessage](#)
- [WinSetClassMsgInterest](#)
- [WinSetMsgInterest](#)
- [WinSetMsgMode](#)
- [WinSetSynchroMode](#)
- [WinTerminate](#)
- [WinWaitMsg](#)

WinCreateMsgQueue - Example Code

This example creates a message queue of default size.

```
#define INCL_WINMESSAGEMGR      /* Window Message functions */
#define INCL_WINWINDOWMGR      /* Window Manager functions */
#include <os2.h>

HAB      hab;                  /* Anchor-block handle      */
HMQ      hmq;                  /* Message queue handle     */
QMSG      qmsg;                /* Message                   */

hab = WinInitialize(0);        /* Initialize PM             */
hmq = WinCreateMsgQueue(hab, 0); /* Create default size queue */

.
.    /* Initialize the windows */
.

/* Get and dispatch the messages from the queue */
while (WinGetMsg(hab, &qmsg, 0, 0, 0))
    WinDispatchMsg(hab, &qmsg);
```

WinCreateMsgQueue - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Errors](#)
- [Remarks](#)

[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinCreateObject

WinCreateObject - Syntax

This function creates an instance of object class *pszClassName*, with title *pszTitle*, and places the icon and title in the location referred to by *pszLocation*.

```
#define INCL_WINWORKPLACE
#include <os2.h>

PSZ      pszClassName;    /* Pointer to class name. */
PSZ      pszTitle;        /* Pointer to initial title of object. */
PSZ      pszSetupString;  /* Pointer to setup string. */
PSZ      pszLocation;     /* Folder location. */
ULONG    ulFlags;         /* Creation flags. */
HOBJECT  rc;              /* Handle to the created object. */

rc = WinCreateObject(pszClassName, pszTitle,
    pszSetupString, pszLocation, ulFlags);
```

WinCreateObject Parameter - pszClassName

pszClassName ([PSZ](#)) - input
Pointer to class name.

A pointer to a zero-terminated string which contains the name of the class of which this object is a member.

WinCreateObject Parameter - pszTitle

pszTitle ([PSZ](#)) - input
Pointer to initial title of object.

A pointer to a zero-terminated string which contains the initial title of the object as it is to appear when displayed on the user interface underneath an icon or on the title bar of an open object.

WinCreateObject Parameter - pszSetupString

pszSetupString ([PSZ](#)) - input
Pointer to setup string.

See the Workplace class definition for WPLaunchPad for a table of setup strings used to customize the Toolbar.

WinCreateObject Parameter - pszLocation

pszLocation ([PSZ](#)) - input
Folder location.

This value can be in any of the following formats:

- Predefined object ids of system folders.

"<WP_NOWHERE>"	The hidden folder.
"<LOCATION_DESKTOP>"	The currently active desktop.
"<WP_OS2SYS>"	The System folder.
"<WP_TEMPS>"	The Templates folder.
"<WP_CONFIG>"	The System Setup folder.
"<WP_START>"	The Startup folder.
"<WP_INFO>"	The Information folder.
"<WP_DRIVES>"	The Drives folder.
- Real name specified as a fully qualified path name.

WinCreateObject Parameter - ulFlags

ulFlags ([ULONG](#)) - input
Creation flags.

This parameter can have one of the following values:

- | | |
|--------------------|--|
| CO_FAILIFEXISTS | No object will be created if an object with the given object ID already exists. This is the default. |
| CO_REPLACEIFEXISTS | If an object with the given ID already exists, the existing object should be replaced. |
| CO_UPDATEIFEXISTS | If an object with the given ID already exists, the existing object should be updated with the new information. |

WinCreateObject Return Value - rc

rc ([HOBJECT](#)) - returns
Handle to the created object.

NULLHANDLE
Error occurred.

Other

A handle to the object created. This handle is persistent and can be used for the [WinSetObjectData](#) and [WinDestroyObject](#) function calls.

WinCreateObject - Parameters

pszClassName ([PSZ](#)) - input
Pointer to class name.

A pointer to a zero-terminated string which contains the name of the class of which this object is a member.

pszTitle ([PSZ](#)) - input
Pointer to initial title of object.

A pointer to a zero-terminated string which contains the initial title of the object as it is to appear when displayed on the user interface underneath an icon or on the title bar of an open object.

pszSetupString ([PSZ](#)) - input
Pointer to setup string.

See the Workplace class definition for WPLaunchPad for a table of setup strings used to customize the Toolbar.

pszLocation ([PSZ](#)) - input
Folder location.

This value can be in any of the following formats:

- Predefined object ids of system folders.

"<WP_NOWHERE>"
"<LOCATION_DESKTOP>"
"<WP_OS2SYS>"
"<WP_TEMP>"
"<WP_CONFIG>"
"<WP_START>"
"<WP_INFO>"
"<WP_DRIVES>"

The hidden folder.
The currently active desktop.
The System folder.
The Templates folder.
The System Setup folder.
The Startup folder.
The Information folder.
The Drives folder.

- Real name specified as a fully qualified path name.

ulFlags ([ULONG](#)) - input
Creation flags.

This parameter can have one of the following values:

CO_FAILIFEXISTS

No object will be created if an object with the given object ID already exists. This is the default.

CO_REPLACEIFEXISTS

If an object with the given ID already exists, the existing object should be replaced.

CO_UPDATEIFEXISTS

If an object with the given ID already exists, the existing object should be updated with the new information.

rc ([HOBJECT](#)) - returns
Handle to the created object.

NULLHANDLE

Error occurred.

Other

A handle to the object created. This handle is persistent and can be used for the [WinSetObjectData](#) and [WinDestroyObject](#) function calls.

WinCreateObject - Remarks

The *pszSetupString* contains a series of "keyname=value" pairs, that change the behavior of the object. "keynames" are separated by semicolons, and "values" are separated by commas.

```
"key=value;key2=value1,value2;"
```

If you want a literal comma or a literal semicolon inside one of your fields you must type the following:

^,	A literal comma.
^;	A literal semicolon.

Each object class documents the keynames and the parameters it expects to see immediately following. See the following keyname-value pairs tables:

- WPColorPalette
- WPDisk
- WPFolder
- WPFontPalette
- WPKeyboard
- WPLaunchPad
- WPPalette
- WPPrinter
- WPRPrinter
- WPProgram
- WPProgramFile
- WPSchemePalette
- WPShadow
- WPObject

Note that ALL parameters have safe defaults, so it is never necessary to pass unnecessary parameters to an object.

For more information about object classes, see the *Workplace Shell Programming Guide*.

Using [HOBJECT](#) for .INI files or files in which an application uses a rename/save/delete sequence is not supported.

WinCreateObject - Related Functions

Related Functions

- [WinDeregisterObjectClass](#)
- [WinDestroyObject](#)
- [WinRegisterObjectClass](#)
- [WinReplaceObjectClass](#)
- [WinSetObjectData](#)

WinCreateObject - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Related Functions](#)

WinCreatePointer

WinCreatePointer - Syntax

This function creates a pointer from a bit map.

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndDesktop; /* Desktop-window handle or HWND_DESKTOP. */
HBITMAP   hbmPointer; /* Bit-map handle from which the pointer image is created. */
BOOL      fPointer; /* Pointer-size indicator. */
LONG      xHotspot; /* x-offset of hot spot within pointer from its lower left corner (in pels). */
LONG      yHotspot; /* y-offset of hot spot within pointer from its lower left corner (in pels). */
HPOINTER  hptr; /* Pointer handle. */

hptr = WinCreatePointer(hwndDesktop, hbmPointer,
                        fPointer, xHotspot, yHotspot);
```

WinCreatePointer Parameter - hwndDesktop

hwndDesktop ([HWND](#)) - input
Desktop-window handle or HWND_DESKTOP.

WinCreatePointer Parameter - hbmPointer

hbmPointer ([HBITMAP](#)) - input
Bit-map handle from which the pointer image is created.

The bit map must be logically divided into two sections vertically, each half representing one of the two images used as the successive drawing masks for the pointer.

For an icon, there are two bit map images. The first half is used for the AND mask and the second half is used for the XOR mask. For details of bit map formats, see

WinCreatePointer Parameter - fPointer

fPointer (BOOL) - input
Pointer-size indicator.

TRUE	The bit map should be stretched (if necessary) to the system pointer dimensions.
FALSE	The bit map should be stretched (if necessary) to the system icon dimensions.

WinCreatePointer Parameter - xHotspot

xHotspot (LONG) - input
x-offset of hot spot within pointer from its lower left corner (in pels).

WinCreatePointer Parameter - yHotspot

yHotspot (LONG) - input
y-offset of hot spot within pointer from its lower left corner (in pels).

WinCreatePointer Return Value - hptr

hptr (HPOINTER) - returns
Pointer handle.

NULLHANDLE	Error
Other	Handle of the newly created pointer.

WinCreatePointer - Parameters

hwndDesktop (HWND) - input
Desktop-window handle or HWND_DESKTOP.

hbmPointer (HBITMAP) - input
Bit-map handle from which the pointer image is created.

The bit map must be logically divided into two sections vertically, each half representing one of the two images used as the successive drawing masks for the pointer.

For an icon, there are two bit map images. The first half is used for the AND mask and the second half is used for the XOR mask. For details of bit map formats, see

fPointer ([BOOL](#)) - input
 Pointer-size indicator.

TRUE The bit map should be stretched (if necessary) to the system pointer dimensions.

FALSE The bit map should be stretched (if necessary) to the system icon dimensions.

xHotspot ([LONG](#)) - input
 x-offset of hot spot within pointer from its lower left corner (in pels).

yHotspot ([LONG](#)) - input
 y-offset of hot spot within pointer from its lower left corner (in pels).

hptr ([HPOINTER](#)) - returns
 Pointer handle.

NULLHANDLE Error

Other Handle of the newly created pointer.

WinCreatePointer - Remarks

A pointer can be created either as a true pointer (at pointer size), or as an icon pointer (at icon size). The latter is useful when using icons as direct-manipulation objects that the user can "pick up" and move about the screen as a means of performing some operation.

See also [WinCreatePointerIndirect](#).

This function makes copies of the supplied bit maps.

WinCreatePointer - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
 An invalid window handle was specified.

PMERR_HBITMAP_BUSY (0x2032)
 An internal bit map busy error was detected. The bit map was locked by one thread during an attempt to access it from another thread.

WinCreatePointer - Related Functions

Related Functions

- [WinCreatePointer](#)
- [WinCreatePointerIndirect](#)
- [WinDestroyPointer](#)
- [WinDrawPointer](#)
- [WinLoadPointer](#)
- [WinQueryPointer](#)
- [WinQueryPointerInfo](#)
- [WinQueryPointerPos](#)

- [WinQuerySysPointer](#)
- [WinQuerySysPointerData](#)
- [WinSetPointer](#)
- [WinSetPointerPos](#)
- [WinSetSysPointerData](#)
- [WinShowPointer](#)

WinCreatePointer - Example Code

This example creates a pointer from a bit map during the creation of the window (WM_CREATE). The bit map (id IDP_BITMAP in the EXE file) is loaded via GpiLoadBitmap.

```
#define INCL_WINPOINTERS          /* Window Pointer Functions */
#define INCL_GPIBITMAPS          /* Graphics bit map Functions */
#include <os2.h>

HPS    hps;          /* presentation-space handle */
HWND   hwnd;         /* window handle */
HPOINTER hptr;       /* bit-map pointer handle */
HBITMAP hbm;         /* bit-map handle */

case WM_CREATE:
    hps = WinBeginPaint(hwnd, NULLHANDLE, NULL);
    hbm = GpiLoadBitmap(hps, 0L, IDP_BITMAP, 64L, 64L);
    WinEndPaint(hps);

    hptr = WinCreatePointer(HWND_DESKTOP, hbm,
                           TRUE, /* use true (system) pointer */
                           0, 0); /* hot spot offset (0,0) */
```

WinCreatePointer - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinCreatePointerIndirect

WinCreatePointerIndirect - Syntax

This function creates a colored pointer or icon from a bit map.

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndDesktop; /* Desktop-window handle or HWND_DESKTOP. */
PPOINTERINFO pptri; /* Pointer information structure. */
HPOINTER  hptr; /* Pointer handle. */

hptr = WinCreatePointerIndirect(hwndDesktop,
                                pptri);
```

WinCreatePointerIndirect Parameter - hwndDesktop

hwndDesktop ([HWND](#)) - input
Desktop-window handle or HWND_DESKTOP.

WinCreatePointerIndirect Parameter - pptri

pptri ([PPOINTERINFO](#)) - input
Pointer information structure.

The fields in this structure must be set before the call is made:

- *fPointer* is set to TRUE if a pointer is being created, or to FALSE if an icon is being created.
- *xHotSpot* *yHotSpot* are set to the relative position in the icon or pointer that is associated with the mouse position.
- *hbmPointer* is a bit map that specifies the AND and XOR masks, as used for black and white pointers and icons.
- *hbmColor* is a color bit map that describes the color content of the pointer or icon.

It is an error if *hbmPointer* is NULLHANDLE. Also, the width of *hbmPointer* must be the same as that of *hbmColor* and the height of *hbmPointer* must be double that of *hbmColor* (to allow for both the AND and the XOR mask).

WinCreatePointerIndirect Return Value - hptr

hptr ([HPOINTER](#)) - returns
Pointer handle.

NULLHANDLE	Error
Other	Handle of the newly created pointer or icon.

WinCreatePointerIndirect - Parameters

hWndDesktop ([HWND](#)) - input
Desktop-window handle or HWND_DESKTOP.

pptri ([PPOINTERINFO](#)) - input
Pointer information structure.

The fields in this structure must be set before the call is made:

- *fPointer* is set to TRUE if a pointer is being created, or to FALSE if an icon is being created.
- *xHotSpot* *yHotSpot* are set to the relative position in the icon or pointer that is associated with the mouse position.
- *hbmPointer* is a bit map that specifies the AND and XOR masks, as used for black and white pointers and icons.
- *hbmColor* is a color bit map that describes the color content of the pointer or icon.

It is an error if *hbmPointer* is NULLHANDLE. Also, the width of *hbmPointer* must be the same as that of *hbmColor* and the height of *hbmPointer* must be double that of *hbmColor* (to allow for both the AND and the XOR mask).

hptr ([HPOINTER](#)) - returns
Pointer handle.

NULLHANDLE

Error

Other

Handle of the newly created pointer or icon.

WinCreatePointerIndirect - Remarks

A pointer can be created either as a true pointer (at pointer size), or as an icon pointer (at icon size). The latter is useful when using icons as direct-manipulation objects that the user can "pick up" and move about the screen as a means of performing some operation (see also [WinCreatePointer](#)).

This function makes copies of the supplied bit maps.

WinCreatePointerIndirect - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_HBITMAP_BUSY (0x2032)
An internal bit map busy error was detected. The bit map was locked by one thread during an attempt to access it from another thread.

WinCreatePointerIndirect - Related Functions

Related Functions

- [WinCreatePointer](#)
- WinCreatePointerIndirect

- [WinDestroyPointer](#)
- [WinDrawPointer](#)
- [WinLoadPointer](#)
- [WinQueryPointer](#)
- [WinQueryPointerInfo](#)
- [WinQueryPointerPos](#)
- [WinQuerySysPointer](#)
- [WinQuerySysPointerData](#)
- [WinSetPointer](#)
- [WinSetPointerPos](#)
- [WinSetSysPointerData](#)
- [WinShowPointer](#)

WinCreatePointerIndirect - Example Code

This example creates a colored pointer from a bit map during the creation of the window (WM_CREATE). The pointer bit map (id IDP_BITMAPPTR in the EXE) and color bit map (id IDP_BITMAPCLR in the EXE file) are loaded via GpiLoadBitmap.

```
#define INCL_WINPOINTERS          /* Window Pointer Functions      */
#define INCL_GPIBITMAPS          /* Graphics bit map Functions  */
#include <os2.h>

HPS    hps;                      /* presentation-space handle    */
HWND   hwnd;                    /* window handle                */
HPOINTER hptr;                  /* bit-map pointer handle       */
HBITMAP hbmPointer;             /* bit-map handle (AND/XOR)     */
HBITMAP hbmColor;              /* bit-map handle (color)       */
POINTERINFO pptriPointerInfo; /* pointer info structure        */

case WM_CREATE:
    hps = WinBeginPaint(hwnd, NULLHANDLE, NULL);
    /* load pointer bit map */
    hbmPointer = GpiLoadBitmap(hps, NULLHANDLE, IDP_BITMAPPTR, 64L, 128L);
    /* load color bit map */
    hbmColor = GpiLoadBitmap(hps, NULLHANDLE, IDP_BITMAPCLR, 64L, 64L);
    WinEndPaint(hps);

    /* initialize POINTERINFO structure */
    pptriPointerInfo.fPointer = TRUE; /* creating pointer */
    pptriPointerInfo.xHotspot = 0; /* x coordinate of hotspot */
    pptriPointerInfo.yHotspot = 0; /* y coordinate of hotspot */
    pptriPointerInfo.hbmPointer = hbmPointer;
    pptriPointerInfo.hbmColor = hbmColor;

    hptr = WinCreatePointerIndirect(HWND_DESKTOP,
                                    &pptriPointerInfo);
```

WinCreatePointerIndirect - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Errors](#)
- [Remarks](#)
- [Example Code](#)
- [Related Functions](#)
- [Glossary](#)

WinCreateShadow

WinCreateShadow - Syntax

This function is specific to OS/2 Version 3.0 or higher.

This function creates a shadow of an object and places it in a specified location.

```
#define INCL_WINWORKPLACE
#include <os2.h>

HOBJECT    hObjectofObject; /* Handle of the object from which the shadow is to be created. */
HOBJECT    hObjectofDest;  /* Handle of the folder into which hObjectofObject is to be placed. */
ULONG      ulReserved;     /* Reserved value; must be 0. */
HOBJECT     rc;             /* Handle of the newly created shadow object. */

rc = WinCreateShadow(hObjectofObject, hObjectofDest,
                    ulReserved);
```

WinCreateShadow Parameter - hObjectofObject

hObjectofObject (**HOBJECT**) - input
Handle of the object from which the shadow is to be created.

WinCreateShadow Parameter - hObjectofDest

hObjectofDest (**HOBJECT**) - input
Handle of the folder into which *hObjectofObject* is to be placed.

WinCreateShadow Parameter - ulReserved

ulReserved (**ULONG**) - input
Reserved value; must be 0.

WinCreateShadow Return Value - rc

rc ([HOBJECT](#)) - returns
Handle of the newly created shadow object.

A return value of NULLHANDLE indicates that either *hObjectofDest* is NULLHANDLE or an object with the same name as *hObjectofObject* exists in the destination folder.

WinCreateShadow - Parameters

hObjectofObject ([HOBJECT](#)) - input
Handle of the object from which the shadow is to be created.

hObjectofDest ([HOBJECT](#)) - input
Handle of the folder into which *hObjectofObject* is to be placed.

ulReserved ([ULONG](#)) - input
Reserved value; must be 0.

rc ([HOBJECT](#)) - returns
Handle of the newly created shadow object.

A return value of NULLHANDLE indicates that either *hObjectofDest* is NULLHANDLE or an object with the same name as *hObjectofObject* exists in the destination folder.

WinCreateShadow - Remarks

Using [HOBJECT](#) for .INI files or files in which an application uses a rename/save/delete sequence is not supported. Its REXX counterpart is SysCreateShadowObject.

WinCreateShadow - Related Functions

Related Functions

- [WinCreateObject](#)
 - [WinDestroyObject](#)
 - [WinMoveObject](#)
 - [WinQueryObjectWindow](#)
 - [WinSaveObject](#)
-

WinCreateShadow - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)

[Related Functions](#)
[Glossary](#)

WinCreateStdWindow

WinCreateStdWindow - Syntax

This function creates a standard window.

```
#define INCL_WINFRAMEMGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND      hwndParent;      /* Parent-window handle. */
ULONG     flStyle;         /* Frame-window style. */
PULONG    pflCreateFlags;  /* Frame-creation flags. */
PSZ       pszClassClient;  /* Client-window class name. */
PSZ       pszTitle;        /* Title-bar text. */
ULONG     flStyleClient;   /* Client-window style. */
HMODULE    Resource;       /* Resource identifier. */
ULONG     ulId;            /* Frame-window identifier. */
PHWND     phwndClient;     /* Client-window handle. */
HWND      hwndFrame;       /* Frame-window handle. */

hwndFrame = WinCreateStdWindow(hwndParent,
                               flStyle, pflCreateFlags, pszClassClient,
                               pszTitle, flStyleClient, Resource,
                               ulId, phwndClient);
```

WinCreateStdWindow Parameter - hwndParent

hwndParent ([HWND](#)) - input
Parent-window handle.

If this parameter is a window handle returned from the [WinQueryDesktopWindow](#) function, or is `HWND_DESKTOP`, a main window is created.

If *hwndParent* is a window handle returned from [WinQueryObjectWindow](#), or is `HWND_OBJECT`, an object window is created.

WinCreateStdWindow Parameter - flStyle

flStyle ([ULONG](#)) - input
Frame-window style.

This is a combination of any of the `WS_*` styles frame styles.

The interpretation of the parameters is affected by the use of all the styles, except for WS_MINIMIZED and WS_MAXIMIZED. These two styles are ignored if they are specified.

WinCreateStdWindow Parameter - pflCreateFlags

pflCreateFlags ([PULONG](#)) - input
Frame-creation flags.

This contains a combination of any of the FCF_* flags. The interpretation of the parameters is affected by the use of these flags; see

WinCreateStdWindow Parameter - pszClassClient

pszClassClient ([PSZ](#)) - input
Client-window class name.

If *pszClassClient* is not a zero-length string, a client window of style *flStyleClient* and class *pszClassClient* is created. *pszClassClient* is either an application specified name as defined by [WinRegisterClass](#) or the name of a preregistered WC_* class; see Preregistered class names are of the form "#nnnnn", where nnnnn is 1 through 5 digits corresponding to the value of the WC_* class name constant.

If *pszClassClient* is NULL, no client area is created.

This parameter can also be specified directly as a WC_* constant.

WinCreateStdWindow Parameter - pszTitle

pszTitle ([PSZ](#)) - input
Title-bar text.

This is ignored if FCF_TITLEBAR (or FCF_STANDARD) is not specified in *pflCreateFlags*.

WinCreateStdWindow Parameter - flStyleClient

flStyleClient ([ULONG](#)) - input
Client-window style.

This is ignored if *pszClassClient* is a zero-length string.

WinCreateStdWindow Parameter - Resource

Resource ([HMODULE](#)) - input
Resource identifier.

This is ignored unless FCF_MENU, FCF_STANDARD, FCF_ACCELTABLE, or FCF_ICON is specified.

NULLHANDLE

Resource definitions are contained in the application .EXE file.

Other

The module handle returned by the DosLoadModule or DosQueryModuleHandle call of the Dynamic Link Library (DLL) containing the resource definitions.

WinCreateStdWindow Parameter - ulld

ulld ([ULONG](#)) - input
Frame-window identifier.

The identifier within the resource definition of the required resource.

It is the responsibility of the application to ensure that all of the resources related to one frame window have the same *ulld* value. It must be greater or equal to 0 and less or equal to 0xFFFF.

WinCreateStdWindow Parameter - phwndClient

phwndClient ([PHWND](#)) - output
Client-window handle.

This is returned if a client window is created.

WinCreateStdWindow Return Value - hwndFrame

hwndFrame ([HWND](#)) - returns
Frame-window handle.

This is NULLHANDLE if no window is created.

WinCreateStdWindow - Parameters

hwndParent ([HWND](#)) - input
Parent-window handle.

If this parameter is a window handle returned from the [WinQueryDesktopWindow](#) function, or is HWND_DESKTOP, a main window is

created.

If *hwndParent* is a window handle returned from [WinQueryObjectWindow](#), or is HWND_OBJECT, an object window is created.

flStyle ([ULONG](#)) - input
Frame-window style.

This is a combination of any of the WS_* styles frame styles.

The interpretation of the parameters is affected by the use of all the styles, except for WS_MINIMIZED and WS_MAXIMIZED. These two styles are ignored if they are specified.

pflCreateFlags ([PULONG](#)) - input
Frame-creation flags.

This contains a combination of any of the FCF_* flags. The interpretation of the parameters is affected by the use of these flags; see

pszClassClient ([PSZ](#)) - input
Client-window class name.

If *pszClassClient* is not a zero-length string, a client window of style *flStyleClient* and class *pszClassClient* is created. *pszClassClient* is either an application specified name as defined by [WinRegisterClass](#) or the name of a preregistered WC_* class; see Preregistered class names are of the form "#nnnnn", where nnnnn is 1 through 5 digits corresponding to the value of the WC_* class name constant.

If *pszClassClient* is NULL, no client area is created.

This parameter can also be specified directly as a WC_* constant.

pszTitle ([PSZ](#)) - input
Title-bar text.

This is ignored if FCF_TITLEBAR (or FCF_STANDARD) is not specified in *pflCreateFlags*.

flStyleClient ([ULONG](#)) - input
Client-window style.

This is ignored if *pszClassClient* is a zero-length string.

Resource ([HMODULE](#)) - input
Resource identifier.

This is ignored unless FCF_MENU, FCF_STANDARD, FCF_ACCELTABLE, or FCF_ICON is specified.

NULLHANDLE

Resource definitions are contained in the application .EXE file.

Other

The module handle returned by the DosLoadModule or DosQueryModuleHandle call of the Dynamic Link Library (DLL) containing the resource definitions.

ulld ([ULONG](#)) - input
Frame-window identifier.

The identifier within the resource definition of the required resource.

It is the responsibility of the application to ensure that all of the resources related to one frame window have the same *ulld* value. It must be greater or equal to 0 and less or equal to 0xFFFF.

phwndClient ([PHWND](#)) - output
Client-window handle.

This is returned if a client window is created.

hwndFrame ([HWND](#)) - returns
Frame-window handle.

This is NULLHANDLE if no window is created.

WinCreateStdWindow - Remarks

The window is created with zero width and depth and positioned at the bottom left of the *hwndParent*, unless FCF_SHELLPOSITION is specified, in which case the size and position are set by the shell. The window can be positioned and sized by use of [WinSetWindowPos](#).

If WS_VISIBLE is set, the frame window is created visible. It is recommended that standard windows that are not main windows are created with WS_VISIBLE not set.

hwndFrame is the window handle of the frame window, that is, the window of class WC_FRAME, and has a parent of *hwndParent*.

The frame window is created with identity *uId*, all the component windows, known as the frame controls, have the standard window identifiers FID_*; see The identifier FID_CLIENT is used for the client area of the window.

It may be necessary to change the *uId* of the frame window after it has been created, so that another frame window can be created with the same resource tables, and still maintain distinct window identities. This can be achieved with the [WinSetWindowUShort](#) call.

Some combinations of frame control flags are valid, but leave visual holes in the frame window. Specifically, if the *pflCreateFlags* parameter specifies any of FCF_SYSMENU, FCF_MINBUTTON, FCF_MAXBUTTON or FCF_MINMAX, but not FCF_TITLEBAR, the area of the top title line between the optional system menu and the minimize/maximize icons is not drawn by the default frame window procedure.

None of the following can be used with WinCreateStdWindow:

- WS_CLIPCHILDREN for the frame style
- WS_CLIPSIBLINGS for the style of the client window or any of the frame control windows
- CS_CLIPSIBLINGS for the class style of the window.

If any of the above are specified, the window is not redrawn correctly. Any style can be used for the children of the client. If it really is required that a client or a frame control is CLIPSIBLINGS, the application must ensure that it is in front of the client and all the other frame controls, for it to be drawn.

WinCreateStdWindow - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_INVALID_FLAG (0x1019)
An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

PMERR_INVALID_INTEGER_ATOM (0x1016)
The specified atom is not a valid integer atom.

PMERR_INVALID_ATOM_NAME (0x1015)
An invalid atom name string was passed.

PMERR_ATOM_NAME_NOT_FOUND (0x1017)
The specified atom name is not in the atom table.

WinCreateStdWindow - Related Functions

Related Functions

- [WinCalcFrameRect](#)
- [WinCreateFrameControls](#)
- [WinCreateStdWindow](#)
- [WinCreateWindow](#)
- [WinDefWindowProc](#)
- [WinDestroyWindow](#)

- [WinQueryClassInfo](#)
- [WinQueryClassName](#)
- [WinRegisterClass](#)
- [WinSubclassWindow](#)

WinCreateStdWindow - Example Code

This example shows a typical initialization function for a window. The function first registers the window class, then calls WinCreateStdWindow to create a standard window and returns immediately if the function fails. Otherwise, it continues on to do other initialization processing. Note: The FCF_STANDARD constant can only be used if you have all the resources in defines. If you use this constant without an accelerator table for example, the function will fail.

```
#define INCL_WINFRAMEMGR          /* Window Frame Functions */
#include <os2.h>
#define IDM_RESOURCE 1
HAB hab;                          /* Anchor-block handle */
CHAR szClassName[] = "Generic"; /* window class name */
HWND hwndClient;                 /* handle to the client */
HWND hwndFrame;                 /* handle to the frame */
PFNWP GenericWndProc;
BOOL GenericInit()
{
    ULONG flStyle;

    flStyle = FCF_STANDARD;
    if (!WinRegisterClass(hab, szClassName, GenericWndProc, 0L, 0))
        return (FALSE);

    hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
        0L, /* frame-window style */
        &flStyle, /* window style */
        szClassName, /* class name */
        "Generic Application", /* window title */
        0L, /* default client style */
        NULLHANDLE, /* resource in executable file */
        IDM_RESOURCE, /* resource id */
        &hwndClient); /* receives client window handle */

    if (!hwndFrame)
        return (FALSE);
    else {
        .
        . /* other initialization code */
        .
    }
}
```

WinCreateStdWindow - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Errors](#)
- [Remarks](#)
- [Example Code](#)
- [Related Functions](#)
- [Glossary](#)

WinCreateSwitchEntry

WinCreateSwitchEntry - Syntax

This function adds an entry to the Window List.

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;           /* Anchor-block handle. */
PSWCNTRL pswctlSwitchData; /* Switch data. */
HSWITCH  hswitchSwitch; /* Handle to the newly created Window List entry. */

hswitchSwitch = WinCreateSwitchEntry(hab,
                                     pswctlSwitchData);
```

WinCreateSwitchEntry Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinCreateSwitchEntry Parameter - pswctlSwitchData

pswctlSwitchData ([PSWCNTRL](#)) - input
Switch data.

Contains information about the newly created Window List entry.

If the field of the [SWCNTRL](#) structure is NULL, the system uses the name under which the application is started. This only applies for OS/2 programs, and only for the first call to this function since the program started. Otherwise, a NULL entry name is invalid.

Leading and trailing blanks are removed from the title, which, if necessary, is also truncated to 60 characters.

If the field of the [SWCNTRL](#) structure is NULLHANDLE, the value used by the system when the program was loaded (if it has been loaded) is substituted.

If the field of the [SWCNTRL](#) structure is 0, the current process ID is used.

If the field of the [SWCNTRL](#) structure is 0, the current session ID is used.

WinCreateSwitchEntry Return Value - hswitchSwitch

hswitchSwitch ([HSWITCH](#)) - returns
Handle to the newly created Window List entry.

There is a system limit to the number of Window List entries. However, this is a large number (several hundred) and is unlikely to be reached in practice since other system limits, such as memory size, are likely to be reached first.

NULLHANDLE
Error occurred
Other
Handle to the newly created Window List entry.

WinCreateSwitchEntry - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

pswctlSwitchData ([PSWCNTRL](#)) - input
Switch data.

Contains information about the newly created Window List entry.

If the field of the [SWCNTRL](#) structure is NULL, the system uses the name under which the application is started. This only applies for OS/2 programs, and only for the first call to this function since the program started. Otherwise, a NULL entry name is invalid.

Leading and trailing blanks are removed from the title, which, if necessary, is also truncated to 60 characters.

If the field of the [SWCNTRL](#) structure is NULLHANDLE, the value used by the system when the program was loaded (if it has been loaded) is substituted.

If the field of the [SWCNTRL](#) structure is 0, the current process ID is used.

If the field of the [SWCNTRL](#) structure is 0, the current session ID is used.

hswitchSwitch ([HSWITCH](#)) - returns
Handle to the newly created Window List entry.

There is a system limit to the number of Window List entries. However, this is a large number (several hundred) and is unlikely to be reached in practice since other system limits, such as memory size, are likely to be reached first.

NULLHANDLE
Error occurred
Other
Handle to the newly created Window List entry.

WinCreateSwitchEntry - Remarks

Both this function and the [WinRemoveSwitchEntry](#) function are not required if the main window is created with the frame creation flags FCF_TASKLIST or FCF_STANDARD, as these styles automatically update the Window List when the main window is created or destroyed, or when its title changes (see also [WinAddSwitchEntry](#)).

WinCreateSwitchEntry - Errors

Possible returns from [WinGetLastError](#)

PMERR_NO_SPACE (0x1201)

The limit on the number of Window List entries has been reached with [WinAddSwitchEntry](#).

PMERR_INVALID_WINDOW (0x1206)

The window specified with a Window List call is not a valid frame window.

PMERR_INVALID_SESSION_ID (0x120B)

The specified session identifier is invalid. Either zero (for the application's own session) or a valid identifier must be specified.

WinCreateSwitchEntry - Related Functions

Related Functions

- [WinAddSwitchEntry](#)
- [WinChangeSwitchEntry](#)
- [WinCreateSwitchEntry](#)
- [WinQuerySessionTitle](#)
- [WinQuerySwitchEntry](#)
- [WinQuerySwitchHandle](#)
- [WinQuerySwitchList](#)
- [WinQueryTaskSizePos](#)
- [WinQueryTaskTitle](#)
- [WinRemoveSwitchEntry](#)
- [WinSwitchToProgram](#)

WinCreateSwitchEntry - Example Code

This example creates a task-list entry for program name "Generic: NEW.APP".

```
#define INCL_WINSWITCHLIST      /* Window Switch List Functions */
#define INCL_WINWINDOWMGR      /* Window Manager Functions */

#include <os2.h>

HSWITCH hswitch;              /* task-list entry handle */
SWCNTRL swctl;                /* switch-control data */
PID pid;                      /* process id */
HAB hab;                      /* anchor-block handle */
HWND hwndFrame;              /* frame handle */

hab = WinQueryAnchorBlock(hwndFrame); /* gets anchor block */
WinQueryWindowProcess(hwndFrame, &pid, NULL); /* gets process id */

/* initialize switch structure */
swctl.hwnd = hwndFrame;        /* window handle */
swctl.hwndIcon = NULLHANDLE;   /* icon handle */
swctl.hprog = NULLHANDLE;      /* program handle */
swctl.idProcess = pid;         /* process identifier */
swctl.idSession = 0;           /* session identifier */
swctl.uchVisibility = SWL_VISIBLE; /* visibility */
swctl.fbJump = SWL_JUMPALE;    /* jump indicator */
strcpy(swctl.szSwttitle, "Generic: NEW.APP"); /* program name */

hswitch = WinCreateSwitchEntry(hab, &swctl);
```

WinCreateSwitchEntry - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinCreateWindow

WinCreateWindow - Syntax

WinCreateWindow creates a new window of class *pszClass* and returns *hwnd*.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndParent;      /* Parent-window handle. */
PSZ     pszClass;        /* Registered-class name. */
PSZ     pszName;         /* Window text. */
ULONG   flStyle;         /* Window style. */
LONG    x;               /* x-coordinate of window position. */
LONG    y;               /* y-coordinate of window position. */
LONG    cx;              /* Width of window, in window coordinates. */
LONG    cy;              /* Height of window, in window coordinates. */
HWND    hwndOwner;       /* Owner-window handle. */
HWND    hwndInsertBehind; /* Sibling-window handle. */
ULONG   id;              /* Window identifier. */
PVOID   pCtlData;        /* Pointer to control data. */
PVOID   pPresParams;     /* Presentation parameters. */
HWND    hwnd;            /* Window handle. */

hwnd = WinCreateWindow(hwndParent, pszClass,
    pszName, flStyle, x, y, cx, cy, hwndOwner,
    hwndInsertBehind, id, pCtlData, pPresParams);
```

WinCreateWindow Parameter - hwndParent

hwndParent (HWND) - input
Parent-window handle.

If *hwndParent* is a desktop window handle, or is `HWND_DESKTOP`, a main window is created.

If *hwndParent* is `HWND_OBJECT`, or is a window handle returned by [WinQueryObjectWindow](#), an object window is created.

WinCreateWindow Parameter - pszClass

pszClass ([PSZ](#)) - input
Registered-class name.

pszClass is either an application-specified name as defined by [WinRegisterClass](#) or the name of a preregistered WC_* class.

This parameter can also be specified directly as a WC_* constant.

WinCreateWindow Parameter - pszName

pszName ([PSZ](#)) - input
Window text.

The actual structure of the data is class-specific. It is usually a null-terminated string that is often displayed in the window.

WinCreateWindow Parameter - flStyle

flStyle ([ULONG](#)) - input
Window style.

WinCreateWindow Parameter - x

x ([LONG](#)) - input
x-coordinate of window position.

The value is in window coordinates relative to the origin of the parent window.

WinCreateWindow Parameter - y

y ([LONG](#)) - input
y-coordinate of window position.

The value is in window coordinates relative to the origin of the parent window.

WinCreateWindow Parameter - cx

cx (**LONG**) - input
Width of window, in window coordinates.

WinCreateWindow Parameter - cy

cy (**LONG**) - input
Height of window, in window coordinates.

WinCreateWindow Parameter - hwndOwner

hwndOwner (**HWND**) - input
Owner-window handle.

Windows that send messages send them to their owner, as defined by this parameter. When an owner window is destroyed, all windows owned by it are also destroyed. The owner window must belong to the current thread.

WinCreateWindow Parameter - hwndInsertBehind

hwndInsertBehind (**HWND**) - input
Sibling-window handle.

This is the sibling window behind which *hwnd* is placed. If this parameter is **HWND_TOP** or **HWND_BOTTOM**, *hwnd* is placed on top of all, or behind all of its siblings. This parameter must be **HWND_TOP**, **HWND_BOTTOM**, or a child of *hwndParent*.

WinCreateWindow Parameter - id

id (**ULONG**) - input
Window identifier.

A value given by the application, that enables specific children of a window to be identified. For example, the controls of a dialog have unique identifiers so that an owner can distinguish which control has notified it. Window identifiers are also used for frame windows. It must be greater or equal to 0 and less or equal to 0xFFFF.

WinCreateWindow Parameter - pCtlData

pCtlData ([PVOID](#)) - input
Pointer to control data.

Pointer to a [Control-Data](#) data structure.

The data referenced by this pointer is class-specific data passed to the window procedure by the [WM_CREATE](#) message.

This parameter MUST be a pointer rather than a long.

The first 2 bytes in the data referenced by the pointer must be the total size of the data referenced by the pointer (for example see the [ENTRYFDATA](#) or the [FRAMECDATA](#) structures). PM requires this information to enable it to ensure that the referenced data is accessible to both 16-bit and 32-bit code.

WinCreateWindow Parameter - pPresParams

pPresParams ([PVOID](#)) - input
Presentation parameters.

This is class-specific presentation data passed to the window procedure by the [WM_CREATE](#) message.

WinCreateWindow Return Value - hwnd

hwnd ([HWND](#)) - returns
Window handle.

NULLHANDLE	Error occurred
Other	Window handle.

WinCreateWindow - Parameters

hwndParent ([HWND](#)) - input
Parent-window handle.

If *hwndParent* is a desktop window handle, or is [HWND_DESKTOP](#), a main window is created.

If *hwndParent* is [HWND_OBJECT](#), or is a window handle returned by [WinQueryObjectWindow](#), an object window is created.

pszClass ([PSZ](#)) - input
Registered-class name.

pszClass is either an application-specified name as defined by [WinRegisterClass](#) or the name of a preregistered [WC_*](#) class.

This parameter can also be specified directly as a [WC_*](#) constant.

pszName ([PSZ](#)) - input
Window text.

The actual structure of the data is class-specific. It is usually a null-terminated string that is often displayed in the window.

flStyle ([ULONG](#)) - input
Window style.

x ([LONG](#)) - input
x-coordinate of window position.

The value is in window coordinates relative to the origin of the parent window.

y ([LONG](#)) - input
y-coordinate of window position.

The value is in window coordinates relative to the origin of the parent window.

cx ([LONG](#)) - input
Width of window, in window coordinates.

cy ([LONG](#)) - input
Height of window, in window coordinates.

hwndOwner ([HWND](#)) - input
Owner-window handle.

Windows that send messages send them to their owner, as defined by this parameter. When an owner window is destroyed, all windows owned by it are also destroyed. The owner window must belong to the current thread.

hwndInsertBehind ([HWND](#)) - input
Sibling-window handle.

This is the sibling window behind which *hwnd* is placed. If this parameter is [HWND_TOP](#) or [HWND_BOTTOM](#), *hwnd* is placed on top of all, or behind all of its siblings. This parameter must be [HWND_TOP](#), [HWND_BOTTOM](#), or a child of *hwndParent*.

id ([ULONG](#)) - input
Window identifier.

A value given by the application, that enables specific children of a window to be identified. For example, the controls of a dialog have unique identifiers so that an owner can distinguish which control has notified it. Window identifiers are also used for frame windows. It must be greater or equal to 0 and less or equal to 0xFFFF.

pCtlData ([PVOID](#)) - input
Pointer to control data.

Pointer to a [Control-Data](#) data structure.

The data referenced by this pointer is class-specific data passed to the window procedure by the [WM_CREATE](#) message.

This parameter MUST be a pointer rather than a long.

The first 2 bytes in the data referenced by the pointer must be the total size of the data referenced by the pointer (for example see the [ENTRYFDATA](#) or the [FRAMECDATA](#) structures). PM requires this information to enable it to ensure that the referenced data is accessible to both 16-bit and 32-bit code.

pPresParams ([PVOID](#)) - input
Presentation parameters.

This is class-specific presentation data passed to the window procedure by the [WM_CREATE](#) message.

hwnd ([HWND](#)) - returns
Window handle.

NULLHANDLE
Error occurred
Other
Window handle.

WinCreateWindow - Remarks

The appearance and behavior of a window are determined by its style, which is the combination of the style established by *pszClass* and *//Style* ORed together. Any of the standard styles *WS_** can be used in addition to any class-specific styles that may be defined.

A window is usually created enabled and invisible. For more information on the initial state of a created window, see the list of the standard window styles.

Messages may be received from other processes or threads when this function is called.

This function sends the [WM_CREATE](#) message to the window procedure of the window being created.

This function sends the [WM_ADJUSTWINDOWPOS](#) message after the [WM_CREATE](#) message, and before the window is displayed (if applicable). The values passed are those given to the `WinCreateWindow` function. If the window has style *WS_VISIBLE*, the window is created visible.

The [WM_SIZE](#) message is not sent by the `WinCreateWindow` function while the window is being created. Any required size processing can be performed during the processing of the [WM_CREATE](#) message.

Because windows are often created with zero height or width and sized later, it is good practice not to perform any size-related processing if the size of the window is zero.

If the `WinCreateWindow` function is called for a window of class *WC_FRAME*, the controls specified by are created but not formatted. The frame is formatted when a [WM_FORMATFRAME](#) message is received but this is not sent during window creation. To cause the frame to format, either a [WM_FORMATFRAME](#) message must be sent, or the window position adjusted using the [WinSetWindowPos](#) function call which sends a [WM_SIZE](#) message if the position or size is changed.

The only limitation to the size and position specified for a window is the number range allowed for the size and position parameters; that is, an application can create windows that are larger than the screen or that are positioned partially or totally off the screen. However, the user interface for manipulation of window sizes and positions is affected if part or all of the window is off the screen.

It is recommended that part of the title bar be left on the screen, if the window has one, to enable the user to move the window around.

When a *WC_MENU* window is created with this call, *pCtiData* is assumed to be a menu template, which is used to create the menu. If *pCtiData* is *NULL*, an empty menu is created.

WinCreateWindow - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_INVALID_FLAG (0x1019)
An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

PMERR_NO_MSG_QUEUE (0x1036)

PMERR_INVALID_PARM (0x1303)
A parameter to the function contained invalid data.

PMERR_INVALID_PARAMETERS (0x1208)
An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32 768 to +32 767 cannot be converted to a [SHORT](#), and a negative number cannot be converted to a [ULONG](#) or [USHORT](#).

WinCreateWindow - Related Functions

Related Functions

- [WinCalcFrameRect](#)

- [WinCreateFrameControls](#)
- [WinCreateStdWindow](#)
- [WinCreateWindow](#)
- [WinDefWindowProc](#)
- [WinDestroyWindow](#)
- [WinQueryClassInfo](#)
- [WinQueryClassName](#)
- [WinRegisterClass](#)
- [WinSubclassWindow](#)

WinCreateWindow - Related Messages

Related Messages

- [WM_ADJUSTWINDOWPOS](#)
- [WM_CREATE](#)
- [WM_FORMATFRAME](#)
- [WM_SIZE](#)

WinCreateWindow - Example Code

This example creates a list box window named "List Box" as a child of the desktop located at (0,0), of size 200x100.

```
#define INCL_WINWINDOWMGR          /* Window Manager Functions */
#define INCL_WINLISTBOXES          /* List Box definitions      */
#include <os2.h>

HWND  hwnd;                        /* Cursor display window    */
ULONG ListBoxId;                  /* Window id                */
                                           /* (supplied by application) */
ULONG flStyle;                    /* Window style             */

flStyle = WS_VISIBLE;             /* Create window visible    */

/* Create button window */
hwnd = WinCreateWindow(HWND_DESKTOP, /* Parent window            */
                      WC_LISTBOX,    /* Class name               */
                      "List Box",    /* Window text              */
                      flStyle,        /* Window style             */
                      0, 0,           /* Position (x,y)           */
                      200, 100,       /* Size (width,height)      */
                      NULLHANDLE,     /* Owner window             */
                      HWND_TOP,       /* Sibling window           */
                      ListBoxId,      /* Window id                */
                      NULL,           /* Control data             */
                      NULL);          /* Pres parameters          */
```

WinCreateWindow - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Errors](#)

[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinDdeInitiate

WinDdeInitiate - Syntax

This function is issued by a client application to one or more other applications, to request initiation of a dynamic data exchange conversation with a national language conversation context.

```
#define INCL_WINDDE /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndClient;    /* Client's window handle. */
PSZ       pszAppName;    /* Application name. */
PSZ       pszTopicName;  /* Topic name. */
PCONVCONTEXT pContext;   /* Conversation context. */
BOOL      rc;            /* Success indicator. */

rc = WinDdeInitiate(hwndClient, pszAppName,
                    pszTopicName, pContext);
```

WinDdeInitiate Parameter - hwndClient

hwndClient ([HWND](#)) - input
Client's window handle.

WinDdeInitiate Parameter - pszAppName

pszAppName ([PSZ](#)) - input
Application name.

This is the name of the desired server application. If it is a zero-length string, any application can respond.

Application names may not contain slashes or backslashes.

WinDdeInitiate Parameter - pszTopicName

pszTopicName ([PSZ](#)) - input
Topic name.

This is the name of the desired topic. If it is a zero-length string, each responding application will respond once for each topic which it can support.

WinDdeInitiate Parameter - pContext

pContext ([PCONVCONTEXT](#)) - input
Conversation context.

WinDdeInitiate Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

- | | |
|-------|---|
| TRUE | Successful completion. The WM_DDE_INITIATE message is successfully sent to all appropriate windows. |
| FALSE | Error occurred. |

WinDdeInitiate - Parameters

hwndClient ([HWND](#)) - input
Client's window handle.

pszAppName ([PSZ](#)) - input
Application name.

This is the name of the desired server application. If it is a zero-length string, any application can respond.

Application names may not contain slashes or backslashes.

pszTopicName ([PSZ](#)) - input
Topic name.

This is the name of the desired topic. If it is a zero-length string, each responding application will respond once for each topic which it can support.

pContext ([PCONVCONTEXT](#)) - input
Conversation context.

rc ([BOOL](#)) - returns
Success indicator.

- | | |
|-------|---|
| TRUE | Successful completion. The WM_DDE_INITIATE message is successfully sent to all appropriate windows. |
| FALSE | |

Error occurred.

WinDdeInitiate - Remarks

This function sends a [WM_DDE_INITIATE](#) message to all top level frame windows. These are windows registered with CS_FRAME, whose parent is the desktop window. No message is sent to object windows.

The WinDdeInitiate function does not return to the client application until all receiving applications have, in sequence, processed the [WM_DDE_INITIATE](#) message, and the client application has received all the corresponding [WM_DDE_INITIATEACK](#) messages (see [WinDdeRespond](#)).

To support DDE conversations between applications running in different memory models (16-bit and 32-bit) it is necessary to process all DDE messages in the application window procedure. The use of the [WinDispatchMsg](#) function ensures that conversion is performed on memory or segment addresses.

WinDdeInitiate - Related Functions

Related Functions

- [WinDdeInitiate](#)
- [WinDdePostMsg](#)
- [WinDdeRespond](#)

WinDdeInitiate - Related Messages

Related Messages

- [WM_DDE_INITIATE](#)
- [WM_DDE_INITIATEACK](#)

WinDdeInitiate - Example Code

This example uses WinDdeInitiate to initiate-during the creation of a client window-a dynamic data exchange (DDE) conversation with any available server applications, asking that the server applications respond for each topic they can support. It also allocates the shared memory that will be used once the conversation is established.

```
#define INCL_WINDDE          /* Window DDE Functions          */
#define INCL_DOSMEMMGR      /* Memory Manager values    */
#include <os2.h>

BOOL fSuccess;              /* success indicator        */
HWND hwndClient;           /* client window            */
char pszAppName[15]="";    /* server application       */
char pszTopicName[15]="";  /* topic                    */
CONVCONTEXT Context;
PDDESTRUCT pddeData;       /* DDE structure            */

case WM_CREATE:
    /* issue DDE initialize call */
    fSuccess = WinDdeInitiate(hwndClient, pszAppName,
```

```

        pszTopicName, &Context);

/* allocate shared memory for conversation data */
DosAllocSharedMem((PVOID)pddeData, "DDESHAREMEM",
    sizeof(DDESTRUCT) + 50,
    PAG_READ | PAG_WRITE | PAG_COMMIT |
    OBJ_GIVEABLE | OBJ_GETTABLE);

```

WinDdeInitiate - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinDdePostMsg

WinDdePostMsg - Syntax

This function is issued by an application to post a message to another application with which it is carrying out a dynamic data exchange conversation with a national language conversation context.

```

#define INCL_WINDDE /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndTo;      /* Window handle of target. */
HWND      hwndFrom;    /* Window handle of originator. */
ULONG     usMsgId;     /* Message identifier. */
PDDESTRUCT pData;      /* Pointer to the DDE control structure being passed. */
ULONG     ulOptions;   /* Options. */
BOOL      rc;          /* Success indicator. */

rc = WinDdePostMsg(hwndTo, hwndFrom, usMsgId,
    pData, ulOptions);

```

WinDdePostMsg Parameter - hwndTo

hwndTo ([HWND](#)) - input
 Window handle of target.

WinDdePostMsg Parameter - hwndFrom

hwndFrom ([HWND](#)) - input
Window handle of originator.

WinDdePostMsg Parameter - usMsgId

usMsgId ([ULONG](#)) - input
Message identifier.

Identifies the message to be posted.

The following messages are valid:

[WM_DDE_ACK](#)
[WM_DDE_ADVISE](#)
[WM_DDE_DATA](#)
[WM_DDE_EXECUTE](#)
[WM_DDE_POKE](#)
[WM_DDE_REQUEST](#)
[WM_DDE_TERMINATE](#)
[WM_DDE_UNADVISE](#).

WinDdePostMsg Parameter - pData

pData ([PDDESTRUCT](#)) - input
Pointer to the DDE control structure being passed.

WinDdePostMsg Parameter - ulOptions

ulOptions ([ULONG](#)) - input
Options.

It must be one of the following values:

DDEPM_RETRY

This controls what happens if the message cannot be posted because the destination queue is full.

If this option is set, then message posting is retried at 1-second intervals, until the message is posted successfully. In this case, this function dispatches any messages in the queue of the application issuing this function, by calling the [WinPeekMsg](#) and [WinDispatchMsg](#) functions in a loop, so that messages sent by other applications can be received. This means that the application can continue to receive DDE messages (or other kinds of messages), while attempting to post DDE messages, thereby preventing deadlock between two applications whose queues are

full and who are both attempting to post a message to each other with this option set.

Applications which rely on inspecting messages prior to issuing the [WinPeekMsg](#) function can either, use the [WinSetHook](#) function and detect the above situation in the invoked hook procedure by testing the MSGF_DDEPOSTMSG value of the *msgf* parameter, or not use this option, in order to avoid the deadlock situation.

If this option is not set, then this function returns FALSE without retrying.

Note: If the message posting fails for any other reason (for example, an invalid window handle is specified), this function returns FALSE even if this option has been selected.

DDEPM_NOFREE

This option prevents the WinDdePostMsg call from freeing the shared memory block passed in on the pData parameter. If this option is used, the caller is responsible for freeing the memory block at some subsequent time (for example, the same memory block could be used in multiple calls to WinDdePostMsg and then freed once at the end of those calls.

If this option is not specified, the DDE structure will be freed.

WinDdePostMsg Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinDdePostMsg - Parameters

hwndTo ([HWND](#)) - input
Window handle of target.

hwndFrom ([HWND](#)) - input
Window handle of originator.

usMsgId ([ULONG](#)) - input
Message identifier.

Identifies the message to be posted.

The following messages are valid:

[WM_DDE_ACK](#)
[WM_DDE_ADVISE](#)
[WM_DDE_DATA](#)
[WM_DDE_EXECUTE](#)
[WM_DDE_POKE](#)
[WM_DDE_REQUEST](#)
[WM_DDE_TERMINATE](#)
[WM_DDE_UNADVISE](#).

pData ([PDDESTRUCT](#)) - input
Pointer to the DDE control structure being passed.

ulOptions ([ULONG](#)) - input

Options.

It must be one of the following values:

DDEPM_RETRY

This controls what happens if the message cannot be posted because the destination queue is full.

If this option is set, then message posting is retried at 1-second intervals, until the message is posted successfully. In this case, this function dispatches any messages in the queue of the application issuing this function, by calling the [WinPeekMsg](#) and [WinDispatchMsg](#) functions in a loop, so that messages sent by other applications can be received. This means that the application can continue to receive DDE messages (or other kinds of messages), while attempting to post DDE messages, thereby preventing deadlock between two applications whose queues are full and who are both attempting to post a message to each other with this option set.

Applications which rely on inspecting messages prior to issuing the [WinPeekMsg](#) function can either, use the [WinSetHook](#) function and detect the above situation in the invoked hook procedure by testing the MSGF_DDEPOSTMSG value of the *msgf* parameter, or not use this option, in order to avoid the deadlock situation.

If this option is not set, then this function returns FALSE without retrying.

Note: If the message posting fails for any other reason (for example, an invalid window handle is specified), this function returns FALSE even if this option has been selected.

DDEPM_NOFREE

This option prevents the WinDdePostMsg call from freeing the shared memory block passed in on the pData parameter. If this option is used, the caller is responsible for freeing the memory block at some subsequent time (for example, the same memory block could be used in multiple calls to WinDdePostMsg and then freed once at the end of those calls.

If this option is not specified, the DDE structure will be freed.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinDdePostMsg - Remarks

To support DDE conversations between applications running in different memory models (16-bit and 32-bit) it is necessary to process all DDE messages in the application window procedure. The use of the [WinDispatchMsg](#) function ensures that conversion is performed on memory or segment addresses.

WinDdePostMsg - Related Functions

Related Functions

- [WinDdeInitiate](#)
- [WinDdePostMsg](#)
- [WinDdeRespond](#)

WinDdePostMsg - Related Messages

Related Messages

- [WM_DDE_ACK](#)
- [WM_DDE_ADVISE](#)
- [WM_DDE_DATA](#)
- [WM_DDE_EXECUTE](#)
- [WM_DDE_POKE](#)
- [WM_DDE_REQUEST](#)
- [WM_DDE_TERMINATE](#)
- [WM_DDE_UNADVISE](#)

WinDdePostMsg - Example Code

This example uses WinDdePostMsg to request a security item from the server once it has received an acknowledgement (via WM_DDEINITIATEACK) to the WinDdeInitiate call. Note the use of the shared memory segment to pass and receive necessary information.

```
#define INCL_WINDDE                /* Window DDE Functions          */
#define INCL_DOSMEMMGR             /* Memory Manager values        */
#include <os2.h>

BOOL fSuccess;                    /* success indicator            */
HWND hwndClient;                  /* client window                */
HWND hwndServer;                  /* server window                */
CHAR pszAppName[15]="";          /* server application           */
CHAR pszTopicName[15]="";        /* topic                        */
HWND hwndTo;                      /* target window                */
HWND hwndFrom;                   /* source window                */
USHORT usMsgId;                   /* message id                   */
BOOL fRetry;                      /* retry indicator              */
CONVCONTEXT Context;
PDDESTRUCT pddeData;             /* DDE structure                */
MRESULT mresReply;               /* message return data          */

case WM_CREATE:
    fSuccess = WinDdeInitiate(hwndClient, pszAppName,
                              pszTopicName, &Context);
    DosAllocSharedMem((PVOID)pddeData, "DDESHAREMEM",
                      sizeof(DDESTRUCT) + 50,
                      PAG_READ | PAG_WRITE | PAG_COMMIT |
                      OBJ_GIVEABLE | OBJ_GETTABLE);

case WM_DDE_INITIATEACK:
    /* issue a request message to DDE partner */
    usMsgId = WM_DDE_REQUEST;

    /* initialize DDE conversation structure */
    pddeData->cbData = sizeof(DDESTRUCT); /* Total length */
    pddeData->fsStatus = DDE_FACK; /* Status - positive ack */
    pddeData->usFormat = DDEFMT_TEXT; /* Data format */
    pddeData->offszItemName = sizeof(DDESTRUCT); /* Offset to item */

    /* set name of item to 'Security', copying the information to
       the shared memory at the end of pddeData */
    strcpy((BYTE *)pddeData + pddeData->offszItem,
           SZDDESYS_ITEM_SECURITY);

    /* Offset to beginning of data (notice additional offset due
       to item information) */

    pddeData->offfabData = sizeof(DDESTRUCT) +
                          strlen(SZDDESYS_ITEM_SECURITY);

    /* set name of item to 'Security', copying the information to
       the shared memory at the end of pddeData */
    strcpy((BYTE *)pddeData + pddeData->offszItem,
           SZDDESYS_ITEM_SECURITY);

    fSuccess = WinDdePostMsg(hwndTo, hwndFrom, usMsgId, pddeData,
                             fRetry);
```

WinDdePostMsg - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinDdeRespond

WinDdeRespond - Syntax

This function is issued by a server application to indicate that it can support a dynamic data exchange conversation on a particular topic with a national language conversation context.

```
#define INCL_WINDDE /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndClient; /* Client's window handle. */
HWND      hwndServer; /* Server's window handle. */
PSZ       pszAppName; /* Application name. */
PSZ       pszTopicName; /* Topic name. */
PCONVCONTEXT pContext; /* Conversation context. */
MRESULT    mresReply; /* Message return data. */

mresReply = WinDdeRespond(hwndClient, hwndServer,
                          pszAppName, pszTopicName, pContext);
```

WinDdeRespond Parameter - hwndClient

hwndClient ([HWND](#)) - input
Client's window handle.

WinDdeRespond Parameter - hwndServer

hwndServer ([HWND](#)) - input
Server's window handle.

If a server application is responding for more than one topic, it must use a different window for each topic.

WinDdeRespond Parameter - pszAppName

pszAppName ([PSZ](#)) - input
Application name.

This is the name of the responding server application. It must not be a zero-length string or null.

Application names may not contain slashes or backslashes.

WinDdeRespond Parameter - pszTopicName

pszTopicName ([PSZ](#)) - input
Topic name.

This is the name of the topic which the server is willing to support. It must not be a zero-length string or null.

WinDdeRespond Parameter - pContext

pContext ([PCONVCONTEXT](#)) - input
Conversation context.

WinDdeRespond Return Value - mresReply

mresReply ([MRESULT](#)) - returns
Message return data.

WinDdeRespond - Parameters

hwndClient ([HWND](#)) - input
Client's window handle.

hwndServer ([HWND](#)) - input
Server's window handle.

If a server application is responding for more than one topic, it must use a different window for each topic.

pszAppName ([PSZ](#)) - input
Application name.

This is the name of the responding server application. It must not be a zero-length string or null.

Application names may not contain slashes or backslashes.

pszTopicName ([PSZ](#)) - input
Topic name.

This is the name of the topic which the server is willing to support. It must not be a zero-length string or null.

pContext ([PCONVCONTEXT](#)) - input
Conversation context.

mresReply ([MRESULT](#)) - returns
Message return data.

WinDdeRespond - Remarks

This function is issued by a server application after receiving a [WM_DDE_INITIATE](#) message that identifies this server application (or indicates that any application can respond), and also either identifies a particular topic which the server can support, or asks for all supported topics (see [WinDdeInitiate](#)). This function sends a [WM_DDE_INITIATEACK](#) message back to the client, that is the sender of the [WM_DDE_INITIATE](#) message.

If the server application can respond, it issues this function once if a specific topic was requested, or once for each topic which it can support, if all supported topics were requested.

A DDE conversation is initiated each time this function is successfully issued. The client is expected to terminate all unwanted conversations. Once a conversation is initiated, it is controlled by the client issuing [WinDdePostMsg](#) functions.

To support DDE conversations between applications running in different memory models (16-bit and 32-bit) it is necessary to process all DDE messages in the application window procedure. The use of the [WinDispatchMsg](#) function ensures that conversion is performed on memory or segment addresses.

WinDdeRespond - Related Functions

Related Functions

- [WinDdeInitiate](#)
- [WinDdePostMsg](#)
- [WinDdeRespond](#)

WinDdeRespond - Related Messages

Related Messages

- [WM_DDE_INITIATE](#)
- [WM_DDE_INITIATEACK](#)

WinDdeRespond - Example Code

This example uses WinDdeRespond to respond to an initiate message (WM_DDEINITIATE) generated by the client window issuing WinDdeInitiate. Here, the server responds as a DDE Server that supports a System topic.

```
#define INCL_WINDDE          /* Window DDE Functions          */
#include <os2.h>

HWND  hwndClient;          /* client window          */
HWND  hwndServer;          /* server window           */
char  pszAppName[15]="DDE Server"; /* server application      */
char  pszTopicName[15]=SZDDESYS_TOPIC; /* topic ('System')       */
MRESULT mresReply;          /* message return data     */
CONVCONTEXT Context;

case WM_DDE_INITIATE:
    mresReply = WinDdeRespond(hwndClient, hwndServer, pszAppName,
                              pszTopicName, &Context);
```

WinDdeRespond - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinDefDlgProc

WinDefDlgProc - Syntax

This function invokes the default dialog procedure with *hwndDlg*, *msg*, *mp1*, and *mp2*.

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND      hwndDlg;          /* Dialog-window handle. */
ULONG     msg;              /* Message identity. */
MPARAM    mp1;              /* Parameter 1. */
MPARAM    mp2;              /* Parameter 2. */
```

```
MRESULT mresReply; /* Message-return data. */  
  
mresReply = WinDefDlgProc(hwndDlg, msg, mp1,  
mp2);
```

WinDefDlgProc Parameter - hwndDlg

hwndDlg ([HWND](#)) - input
Dialog-window handle.

WinDefDlgProc Parameter - msg

msg ([ULONG](#)) - input
Message identity.

WinDefDlgProc Parameter - mp1

mp1 ([MPARAM](#)) - input
Parameter 1.

WinDefDlgProc Parameter - mp2

mp2 ([MPARAM](#)) - input
Parameter 2.

WinDefDlgProc Return Value - mresReply

mresReply ([MRESULT](#)) - returns
Message-return data.

WinDefDlgProc - Parameters

hwndDlg ([HWND](#)) - input
Dialog-window handle.

msg ([ULONG](#)) - input
Message identity.

mp1 ([MPARAM](#)) - input
Parameter 1.

mp2 ([MPARAM](#)) - input
Parameter 2.

mresReply ([MRESULT](#)) - returns
Message-return data.

WinDefDlgProc - Remarks

The action taken by the default dialog procedure is such that the values passed in *mp1* and *mp2*, and the values returned in *mresReply* are defined for each *msg*.

The default dialog procedure provides default processing for any dialog window messages that an application chooses not to process. It can be used to ensure that every message is processed and is called with the same parameters that were received by the dialog procedure.

The action of the WinDefDlgProc function on receiving messages is precisely the same as for the frame window procedure except for WM_CLOSE messages where WinDismissDlg will be called. If an application processes a message instead of sending it to the WinDefDlgProc function, it may be required to perform some or all of the frame window procedure actions for itself.

WinDefDlgProc - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinDefDlgProc - Related Functions

Related Functions

- [WinCreateDlg](#)
- [WinDefDlgProc](#)
- [WinDismissDlg](#)
- [WinDlgBox](#)
- [WinGetDlgMsg](#)
- [WinLoadDlg](#)
- [WinProcessDlg](#)

WinDefDlgProc - Example Code

This example shows a typical dialog box procedure. A switch statement is used to process individual messages. All messages not processed are passed on to the WinDefDlgProc function.

```
#define INCL_WINDIALOGS          /* Window Dialog Mgr Functions */
#include <os2.h>

MRESULT AboutDlg(HWND hwnd, ULONG ulMessage, MPARAM mpParam1,
                  MPARAM mpParam2)
{
    switch (ulMessage) {

        /*
         * Process whatever messages you want here and send the rest
         * to WinDefDlgProc.
         */

        default:
            return (WinDefDlgProc(hwnd, ulMessage, mpParam1, mpParam2));
    }
}
```

WinDefDlgProc - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinDefFileDlgProc

WinDefFileDlgProc - Syntax

This function is the default dialog procedure for the file dialog.

```
#define INCL_winstdfile
#include <os2.h>

HWND      hwnd;          /* Dialog-window handle. */
ULONG     msg;           /* Message identity. */
MPARAM    mp1;           /* Parameter 1. */
MPARAM    mp2;           /* Parameter 2. */
MRESULT   mresReply;     /* Message-return data. */

mresReply = WinDefFileDlgProc(hwnd, msg, mp1,
                               mp2);
```

WinDefFileDlgProc Parameter - hwnd

hwnd ([HWND](#)) - input
Dialog-window handle.

WinDefFileDlgProc Parameter - msg

msg ([ULONG](#)) - input
Message identity.

WinDefFileDlgProc Parameter - mp1

mp1 ([MPARAM](#)) - input
Parameter 1.

WinDefFileDlgProc Parameter - mp2

mp2 ([MPARAM](#)) - input
Parameter 2.

WinDefFileDlgProc Return Value - mresReply

mresReply ([HRESULT](#)) - returns
Message-return data.

WinDefFileDlgProc - Parameters

hwnd ([HWND](#)) - input

Dialog-window handle.

msg ([ULONG](#)) - input
Message identity.

mp1 ([MPARAM](#)) - input
Parameter 1.

mp2 ([MPARAM](#)) - input
Parameter 2.

mresReply ([MRESULT](#)) - returns
Message-return data.

WinDefFileDlgProc - Remarks

All unprocessed messages in a custom dialog procedure should be passed to the default file dialog procedure so that the dialog can implement its default behavior.

WinDefFileDlgProc - Example Code

This example uses the default dialog procedure for the file dialog to cause default processing of unprocessed dialog messages.

```
#define INCL_WINSTDFILE  /* Window Standard File Functions      */
#include <os2.h>

MRESULT MyFileDlgProc(HWND hwnd, ULONG msg, MPARAM mpParam1,
                      MPARAM mpParam2)
{
    switch(msg)
    {
        /******
        /* Process user-supported messages
        /******
        .
        .
        .
        default:
            return (WinDefFileDlgProc(hwnd, msg, mpParam1, mpParam2));
    }
}
```

WinDefFileDlgProc - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

WinDefFontDlgProc

WinDefFontDlgProc - Syntax

This function is the default dialog procedure for the font dialog.

```
#define INCL_winstdfont
#include <os2.h>

HWND      hwnd;      /* Dialog-window handle. */
ULONG     msg;        /* Message identity. */
MPARAM    mp1;        /* Parameter 1. */
MPARAM    mp2;        /* Parameter 2. */
MRESULT    mresReply; /* Message-return data. */

mresReply = WinDefFontDlgProc(hwnd, msg, mp1,
                               mp2);
```

WinDefFontDlgProc Parameter - hwnd

hwnd (**HWND**) - input
Dialog-window handle.

WinDefFontDlgProc Parameter - msg

msg (**ULONG**) - input
Message identity.

WinDefFontDlgProc Parameter - mp1

mp1 (**MPARAM**) - input
Parameter 1.

WinDefFontDlgProc Parameter - mp2

mp2 (**MPARAM**) - input
Parameter 2.

WinDefFontDlgProc Return Value - mresReply

mresReply (**MRESULT**) - returns
Message-return data.

WinDefFontDlgProc - Parameters

hwnd (**HWND**) - input
Dialog-window handle.

msg (**ULONG**) - input
Message identity.

mp1 (**MPARAM**) - input
Parameter 1.

mp2 (**MPARAM**) - input
Parameter 2.

mresReply (**MRESULT**) - returns
Message-return data.

WinDefFontDlgProc - Remarks

All unprocessed messages in a custom dialog procedure should be passed to the default font dialog procedure so that the dialog can implement its default behavior.

WinDefFontDlgProc - Example Code

This example uses the default dialog procedure for the font dialog to cause default processing of unprocessed dialog messages.

```
#define INCL_WINSTDFONT /* Window Standard Font Functions */
#include <os2.h>

MRESULT MyFontDlgProc(HWND hwnd, ULONG msg, MPARAM mpParam1,
                      MPARAM mpParam2)
{
    switch(msg)
    {
        /*****
        *****/
```

```

/* Process user-supported messages */
/*****
.
.
.
default:
    return (WinDefFontDlgProc(hwnd, msg, mpParam1, mpParam2));
}
}

```

WinDefFontDlgProc - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

WinDefWindowProc

WinDefWindowProc - Syntax

This function invokes the default window procedure.

```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND      hwnd;      /* Window handle. */
ULONG     ulMsgid;    /* Message identity. */
MPARAM    mpParam1;   /* Parameter 1. */
MPARAM    mpParam2;   /* Parameter 2. */
MRESULT   mresReply;  /* Message-return data. */

mresReply = WinDefWindowProc(hwnd, ulMsgid,
                             mpParam1, mpParam2);

```

WinDefWindowProc Parameter - hwnd

hwnd (**HWND**) - input
Window handle.

WinDefWindowProc Parameter - ulMsgid

ulMsgid ([ULONG](#)) - input
Message identity.

WinDefWindowProc Parameter - mpParam1

mpParam1 ([MPARAM](#)) - input
Parameter 1.

WinDefWindowProc Parameter - mpParam2

mpParam2 ([MPARAM](#)) - input
Parameter 2.

WinDefWindowProc Return Value - mresReply

mresReply ([MRESULT](#)) - returns
Message-return data.

WinDefWindowProc - Parameters

hwnd ([HWND](#)) - input
Window handle.

ulMsgid ([ULONG](#)) - input
Message identity.

mpParam1 ([MPARAM](#)) - input
Parameter 1.

mpParam2 ([MPARAM](#)) - input
Parameter 2.

mresReply ([MRESULT](#)) - returns
Message-return data.

WinDefWindowProc - Remarks

The default window provides default processing for any window messages that an application chooses not to process. It can be used to ensure that every message is processed. This function should be made with the same parameters as those received by the window procedure.

The action taken by the default window procedure, the values passed in *mpParam1*, *mpParam2* and the values returned in *mresReply* are defined for each *ulMsgid*.

WinDefWindowProc - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinDefWindowProc - Related Functions

Related Functions

- [WinCalcFrameRect](#)
 - [WinCreateFrameControls](#)
 - [WinCreateStdWindow](#)
 - [WinCreateWindow](#)
 - [WinDefWindowProc](#)
 - [WinDestroyWindow](#)
 - [WinQueryClassInfo](#)
 - [WinQueryClassName](#)
 - [WinRegisterClass](#)
 - [WinSubclassWindow](#)
-

WinDefWindowProc - Example Code

This example uses the default window procedure, called by WinDefWindowProc, for default processing of non supported window messages.

```
#define INCL_WINWINDOWMGR          /* Window Manager Functions */
#include <os2.h>

MRESULT GenericWndProc(HWND hwnd, ULONG ulMsgid, MPARAM mpParam1,
                        MPARAM mpParam2)
{
    switch(ulMsgid)
    {
        /*
         * process user supported messages
         */

        default:
            return (WinDefWindowProc(hwnd, ulMsgid, mpParam1, mpParam2));
    }
}
```

```
}  
}
```

WinDefWindowProc - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinDeleteAtom

WinDeleteAtom - Syntax

This function deletes an atom from an atom table.

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */  
#include <os2.h>  
  
HATOMTBL    hatomtblAtomTbl; /* Atom-table handle. */  
ATOM         atom;           /* Atom identifying the atom to be deleted. */  
ATOM         rc;             /* Return code. */  
  
rc = WinDeleteAtom(hatomtblAtomTbl, atom);
```

WinDeleteAtom Parameter - hatomtblAtomTbl

hatomtblAtomTbl ([HATOMTBL](#)) - input
Atom-table handle.

This is the handle returned from a previous [WinCreateAtomTable](#) or [WinQuerySystemAtomTable](#) function.

WinDeleteAtom Parameter - atom

atom ([ATOM](#)) - input
Atom identifying the atom to be deleted.

WinDeleteAtom Return Value - rc

rc ([ATOM](#)) - returns
Return code.

0	Call successful
Other	The call fails and the atom has not been deleted, in which case this is equal to the <i>atom</i> parameter.

WinDeleteAtom - Parameters

hatomtblAtomTbl ([HATOMTBL](#)) - input
Atom-table handle.

This is the handle returned from a previous [WinCreateAtomTable](#) or [WinQuerySystemAtomTable](#) function.

atom ([ATOM](#)) - input
Atom identifying the atom to be deleted.

rc ([ATOM](#)) - returns
Return code.

0	Call successful
Other	The call fails and the atom has not been deleted, in which case this is equal to the <i>atom</i> parameter.

WinDeleteAtom - Remarks

If the passed atom is an integer atom, 0 is returned. If it is not an integer atom and it is a valid atom for the given atom table, that is, it has an atom name and use count, its use count is decremented by one and 0 is returned. If the use count has been decremented to zero, the atom name and use count are removed from the atom table.

WinDeleteAtom - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HATOMTBL (0x1013)
An invalid atom-table handle was specified.

PMERR_INVALID_ATOM (0x1014)

The specified atom does not exist in the atom table.

WinDeleteAtom - Related Functions

Related Functions

- [WinAddAtom](#)
- [WinCreateAtomTable](#)
- [WinDeleteAtom](#)
- [WinDestroyAtomTable](#)
- [WinFindAtom](#)
- [WinQueryAtomLength](#)
- [WinQueryAtomName](#)
- [WinQueryAtomUsage](#)
- [WinQuerySystemAtomTable](#)

WinDeleteAtom - Example Code

This example deletes a newly created atom in an Atom Table based on the atom value returned by WinAddAtom.

```
#define INCL_WINATOM           /* Window Atom Functions          */
#include <os2.h>

ATOM  atom;                    /* new atom value              */
ATOM  atomDelete;              /* result of atom delete       */
HATOMTBL hatomtblAtomTbl; /* atom-table handle           */
char   pszAtomName[10]; /* atom name                   */
ULONG  ulInitial = 0; /* initial atom table size (use default) */
ULONG  ulBuckets = 0; /* size of hash table (use default) */

/* create atom table of default size */
hatomtblAtomTbl = WinCreateAtomTable(ulInitial, ulBuckets);

/* define name for new atom and add to table */
strcpy(pszAtomName, "newatom");
atom = WinAddAtom(hatomtblAtomTbl, pszAtomName);

atomDelete = WinDeleteAtom(hatomtblAtomTbl, atom);
```

WinDeleteAtom - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinDeleteLboxItem

WinDeleteLboxItem - Syntax

This macro deletes the indexed item from the List Box. It returns the number of items left.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndLbox; /* Listbox handle. */
LONG    index;    /* Index of the listbox item. */
LONG    lItems;   /* Number of items left. */

lItems = WinDeleteLboxItem(hwndLbox, index);
```

WinDeleteLboxItem Parameter - hwndLbox

hwndLbox (HWND) - input
Listbox handle.

WinDeleteLboxItem Parameter - index

index (LONG) - input
Index of the listbox item.

WinDeleteLboxItem Return Value - lItems

lItems (LONG) - returns
Number of items left.

WinDeleteLboxItem - Parameters

hwndLbox ([HWND](#)) - input
Listbox handle.

index ([LONG](#)) - input
Index of the listbox item.

lItems ([LONG](#)) - returns
Number of items left.

WinDeleteLboxItem - Remarks

This macro is defined as:

```
#define WinDeleteLboxItem(hwndLbox, index) \
    ((LONG)WinSendMessage(hwndLbox, \
        LM_DELETEITEM, \
        MPFROMLONG(index), \
        (MPARAM)NULL))
```

This macro requires the existence of a message queue.

WinDeleteLboxItem - Related Functions

Related Functions

- [WinSendMessage](#)
-

WinDeleteLboxItem - Related Messages

Related Messages

- [LM_DELETEITEM](#)
-

WinDeleteLboxItem - Example Code

This example responds to an item in the list box being selected (LN_SELECT, WM_CONTROL message) by deleting the selected item using WinDeleteLboxItem.

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#define INCL_WINLISTBOXES     /* Window List Box definitions */
#include <os2.h>

LONG    lIndex;                /* selected item index */
LONG    lLeft;                 /* items left after delete */
HWND    hwndLbox;              /* list box window handle */
MPARAM  mpParam1;              /* Parameter 1 (rectl structure) */
MPARAM  mpParam2;              /* Parameter 2 (frame boolean) */
```

```

case WM_CONTROL:
    /* switch on control code */
    switch(SHORT2FROMMP(mpParam1))
    {
        case LN_SELECT:
            hwndLbox = HWNDFROMMP(mpParam2);

            /* query index of selected item */
            lIndex = WinQueryLboxSelectedItem(hwndLbox);

            /* delete selected listbox item */
            lLeft = WinDeleteLboxItem(hwndLbox, lIndex);
            break;
    }

```

WinDeleteLboxItem - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Related Messages](#)

[Glossary](#)

WinDeleteLibrary

WinDeleteLibrary - Syntax

This function deletes the library *hlibLibhandle*, which is previously loaded by the [WinLoadLibrary](#) function.

```

#define INCL_WINLOAD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
HLIB     hlibLibhandle; /* Library handle to be deleted. */
BOOL     rc;           /* Library-deleted indicator. */

rc = WinDeleteLibrary(hab, hlibLibhandle);

```

WinDeleteLibrary Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinDeleteLibrary Parameter - hlibLibhandle

hlibLibhandle ([HLIB](#)) - input
Library handle to be deleted.

This handle was previously loaded by the the [WinLoadLibrary](#) function.

WinDeleteLibrary Return Value - rc

rc ([BOOL](#)) - returns
Library-deleted indicator.

TRUE	Library successfully deleted
FALSE	Library not successfully deleted.

WinDeleteLibrary - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

hlibLibhandle ([HLIB](#)) - input
Library handle to be deleted.

This handle was previously loaded by the the [WinLoadLibrary](#) function.

rc ([BOOL](#)) - returns
Library-deleted indicator.

TRUE	Library successfully deleted
FALSE	Library not successfully deleted.

WinDeleteLibrary - Related Functions

Related Functions

- [WinDeleteLibrary](#)
- [WinDeleteProcedure](#)
- [WinLoadLibrary](#)

- [WinLoadProcedure](#)

WinDeleteLibrary - Example Code

This example deletes the library identified by the library handle returned from WinLoadLibrary.

```
#define INCL_WINLOAD          /* Window Load Functions      */
#include <os2.h>

BOOL    fSuccess;           /* success indicator      */
HAB     hab;                /* anchor-block handle    */
HLIB    hlib;               /* library handle         */

fSuccess = WinDeleteLibrary(hab, hlib);
```

WinDeleteLibrary - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinDeleteProcedure

WinDeleteProcedure - Syntax

This function deletes the window or dialog procedure that was previously loaded using the [WinLoadProcedure](#) function.

```
#define INCL_WINLOAD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB     hab;              /* Anchor-block handle. */
PFNWP   pwndproc;        /* Window procedure identifier to be deleted. */
BOOL     rc;              /* Procedure-deleted indicator. */

rc = WinDeleteProcedure(hab, pwndproc);
```

WinDeleteProcedure Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinDeleteProcedure Parameter - pwndproc

pwndproc ([PFNWP](#)) - input
Window procedure identifier to be deleted.

WinDeleteProcedure Return Value - rc

rc ([BOOL](#)) - returns
Procedure-deleted indicator.

TRUE	Procedure successfully deleted
FALSE	Procedure not successfully deleted.

WinDeleteProcedure - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

pwndproc ([PFNWP](#)) - input
Window procedure identifier to be deleted.

rc ([BOOL](#)) - returns
Procedure-deleted indicator.

TRUE	Procedure successfully deleted
FALSE	Procedure not successfully deleted.

WinDeleteProcedure - Related Functions

Related Functions

- [WinDeleteLibrary](#)
- [WinDeleteProcedure](#)
- [WinLoadLibrary](#)
- [WinLoadProcedure](#)

WinDeleteProcedure - Example Code

This example deletes the procedure identified by the procedure pointer returned from WinLoadProcedure.

```
#define INCL_WINLOAD          /* Window Load Functions */
#include <os2.h>

BOOL    fSuccess;           /* success indicator */
PFNWP   pWndproc;           /* procedure pointer */
HAB     hab;                /* anchor-block handle */

fSuccess = WinDeleteProcedure(hab, pWndproc);
```

WinDeleteProcedure - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinDeregisterObjectClass

WinDeregisterObjectClass - Syntax

This function deregisters (removes) a workplace object class.

```
#define INCL_WINWORKPLACE
#include <os2.h>

PSZ     pszClassName;      /* Pointer to class name. */
BOOL    rc;                /* Success indicator. */

rc = WinDeregisterObjectClass(pszClassName);
```

WinDeregisterObjectClass Parameter - pszClassName

pszClassName ([PSZ](#)) - input
Pointer to class name.

A pointer to a zero-terminated string which contains the name of the object class being removed from the workplace.

WinDeregisterObjectClass Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinDeregisterObjectClass - Parameters

pszClassName ([PSZ](#)) - input
Pointer to class name.

A pointer to a zero-terminated string which contains the name of the object class being removed from the workplace.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinDeregisterObjectClass - Remarks

Workplace object classes are not deleted unless the application issues a WinDeregisterObjectClass. Object classes will be automatically registered when a dynamic-link library containing an object definition is added to the system. The only advantage of deregistering an object class is to optimize the system performance. All registered classes are maintained in the OS2.INI and are cached upon system initialization. If the class is no longer needed, it should be removed.

WinDeregisterObjectClass - Related Functions

Related Functions

- [WinCreateObject](#)
- [WinRegisterObjectClass](#)
- [WinReplaceObjectClass](#)

WinDeregisterObjectClass - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

WinDestroyAccelTable

WinDestroyAccelTable - Syntax

This function destroys an accelerator table.

```
#define INCL_WINACCELERATORS /* Or use INCL_WIN, INCL_PM, */  
#include <os2.h>  
  
HACCEL    haccelAccel; /* Accelerator-table handle. */  
BOOL      rc;          /* Success indicator. */  
  
rc = WinDestroyAccelTable(haccelAccel);
```

WinDestroyAccelTable Parameter - haccelAccel

haccelAccel ([HACCEL](#)) - input
Accelerator-table handle.

WinDestroyAccelTable Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinDestroyAccelTable - Parameters

haccelAccel ([HACCEL](#)) - input
Accelerator-table handle.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinDestroyAccelTable - Remarks

Before an application is terminated, it should call the WinDestroyAccelTable function for every accelerator table that is created with the [WinCreateAccelTable](#) function.

WinDestroyAccelTable - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HACCEL (0x101A)
An invalid accelerator-table handle was specified.

WinDestroyAccelTable - Related Functions

Related Functions

- [WinCopyAccelTable](#)
- [WinCreateAccelTable](#)
- [WinDestroyAccelTable](#)
- [WinLoadAccelTable](#)
- [WinQueryAccelTable](#)
- [WinSetAccelTable](#)
- [WinTranslateAccel](#)

WinDestroyAccelTable - Example Code

This example destroys an accelerator-table based on the handle returned from either [WinCreateAccelTable](#) or [WinLoadAccelTable](#).

```

#define INCL_WINACCELERATORS    /* Window Accelerator Functions */
#include <os2.h>

HACCEL  hAccel;                /* Accelerator-table handle      */
BOOL    fSuccess;              /* success indicator             */

fSuccess = WinDestroyAccelTable(hAccel);

```

WinDestroyAccelTable - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinDestroyAtomTable

WinDestroyAtomTable - Syntax

This function destroys a private atom table, which is created by [WinCreateAtomTable](#).

```

#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HATOMTBL    hatomtblAtomTbl; /* Atom-table handle. */
HATOMTBL    rc;              /* Return code. */

rc = WinDestroyAtomTable(hatomtblAtomTbl);

```

WinDestroyAtomTable Parameter - hatomtblAtomTbl

hatomtblAtomTbl ([HATOMTBL](#)) - input
Atom-table handle.

This is the handle returned from a previous call to the [WinCreateAtomTable](#) function. If NULL then this function does nothing.

WinDestroyAtomTable Return Value - rc

rc (HATOMTBL) - returns	Return code.
0	Function successful.
Other	The call fails and the atom table has not been destroyed, in which case this is equal to the <i>hatomtblAtomTbl</i> parameter.

WinDestroyAtomTable - Parameters

hatomtblAtomTbl (HATOMTBL) - input	Atom-table handle.
This is the handle returned from a previous call to the WinCreateAtomTable function. If NULL then this function does nothing.	
rc (HATOMTBL) - returns	Return code.
0	Function successful.
Other	The call fails and the atom table has not been destroyed, in which case this is equal to the <i>hatomtblAtomTbl</i> parameter.

WinDestroyAtomTable - Remarks

This function makes no attempt to ensure that the handle to the atom table is not reused by a later call to the [WinCreateAtomTable](#) function.

The system atom table (see the [WinQuerySystemAtomTable](#) function) cannot be destroyed.

WinDestroyAtomTable - Errors

Possible returns from WinGetLastError
PMERR_INVALID_HATOMTBL (0x1013)
An invalid atom-table handle was specified.

WinDestroyAtomTable - Related Functions

Related Functions

- [WinAddAtom](#)
- [WinCreateAtomTable](#)
- [WinDeleteAtom](#)
- [WinDestroyAtomTable](#)
- [WinFindAtom](#)
- [WinQueryAtomLength](#)
- [WinQueryAtomName](#)
- [WinQueryAtomUsage](#)
- [WinQuerySystemAtomTable](#)

WinDestroyAtomTable - Example Code

This example destroys an Atom Table of one atom, based on its handle, which is returned by WinCreateAtomTable.

```
#define INCL_WINATOM                /* Window Atom Functions          */
#include <os2.h>

ATOM atom;                          /* new atom value                */
HATOMTBL hatomtblAtomTbl; /* atom-table handle            */
HATOMTBL hatomtblDestroy; /* result of destroy table      */
char pszAtomName[10]; /* atom name                    */
USHORT usInitial = 0; /* initial atom table size (use default)*/
USHORT usBuckets = 0; /* size of hash table (use default) */

/* create atom table of default size */
hatomtblAtomTbl = WinCreateAtomTable(usInitial, usBuckets);

/* define name for new atom and add to table */
strcpy(pszAtomName, "newatom");
atom = WinAddAtom(hatomtblAtomTbl, pszAtomName);

hatomtblDestroy = WinDestroyAtomTable(hatomtblAtomTbl);
```

WinDestroyAtomTable - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Errors](#)
- [Remarks](#)
- [Example Code](#)
- [Related Functions](#)
- [Glossary](#)

WinDestroyCursor

WinDestroyCursor - Syntax

This function destroys the current cursor, if it belongs to the specified window.

```
#define INCL_WINCursors /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND    hwnd; /* Window handle to which the cursor belongs. */
BOOL    rc;    /* Success indicator. */

rc = WinDestroyCursor(hwnd);
```

WinDestroyCursor Parameter - hwnd

hwnd (**HWND**) - input
Window handle to which the cursor belongs.

WinDestroyCursor Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinDestroyCursor - Parameters

hwnd (**HWND**) - input
Window handle to which the cursor belongs.

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinDestroyCursor - Remarks

This function has no effect if the current cursor does not belong to the specified window.

It is not necessary to call this function before calling the [WinCreateCursor](#) function.

If the cursor was created with the CURSOR_FLASH option, then the TID_CURSOR timer is also destroyed.

WinDestroyCursor - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinDestroyCursor - Related Functions

Related Functions

- [WinCreateCursor](#)
- [WinDestroyCursor](#)
- [WinQueryCursorInfo](#)
- [WinShowCursor](#)

WinDestroyCursor - Example Code

This example destroys the cursor defined for the specified input window.

```
#define INCL_WINCursors          /* Window Cursor Functions      */
#include <os2.h>

BOOL fSuccess;                  /* success indicator          */
HWND hwnd;                      /* cursor display window      */

fSuccess = WinDestroyCursor(hwnd);
```

WinDestroyCursor - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinDestroyHelpInstance

WinDestroyHelpInstance - Syntax

This function destroys the specified instance of the Help Manager.

```
#define INCL_WINHELP /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndHelpInstance; /* Handle of the instance of the Help Manager to be destroyed. */
BOOL    rc;                /* Success indicator. */

rc = WinDestroyHelpInstance(hwndHelpInstance);
```

WinDestroyHelpInstance Parameter - hwndHelpInstance

hwndHelpInstance ([HWND](#)) - input
Handle of the instance of the Help Manager to be destroyed.

This is the handle returned by the [WinCreateHelpInstance](#) call.

WinDestroyHelpInstance Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinDestroyHelpInstance - Parameters

hwndHelpInstance ([HWND](#)) - input
Handle of the instance of the Help Manager to be destroyed.

This is the handle returned by the [WinCreateHelpInstance](#) call.

rc ([BOOL](#)) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinDestroyHelpInstance - Related Functions

Related Functions

- [WinAssociateHelpInstance](#)
- [WinCreateHelpInstance](#)
- [WinCreateHelpTable](#)
- [WinDestroyHelpInstance](#)
- [WinLoadHelpTable](#)
- [WinQueryHelpInstance](#)

WinDestroyHelpInstance - Example Code

This example shows a typical main function for an application which uses help. Following creation of the main application window the Help Manager is initialized and associated with the window. The help table is defined in the application's resources. When the window is destroyed, terminating the application, the help instance is also destroyed.

```
#define INCL_WIN
#include <os2.h>

#define IDHT_APPLICATION      100      /* id of HELP TABLE in resource file
*/

main( int argc, char *argv[], char *envp[] )
{
    HAB hab = WinInitialize( 0 );
    HMQ hmq = WinCreateMsgQueue( hab, 0 );
    HWND hwnd;
    HWND hwndClient;
    HWND hwndHelp;
    QMSG qmsg;
    ULONG flStyle;
    HELPINIT helpinit;

    /* Setup the help initialization structure */
    helpinit.cb = sizeof( HELPINIT );
    helpinit.ulReturnCode = 0L;
    helpinit.pszTutorialName = (PSZ)NULL;
    /* Help table in application resource */
    helpinit.phtHelpTable = (PHELPTABLE)MAKEULONG( IDHT_APPLICATION, 0xffff );
    helpinit.hmodHelpTableModule = NULLHANDLE;
    /* Default action bar and accelerators */
    helpinit.hmodAccelActionBarModule = NULLHANDLE;
    helpinit.idAccelTable = 0;
    helpinit.idActionBar = 0;
    helpinit.pszHelpWindowTitle = "APPNAME HELP";
    helpinit.fShowPanelId = CMIC_SHOW_PANEL_ID;
    helpinit.pszHelpLibraryName = "APPNAME.HLP";

    /* Register the class */
    if( WinRegisterClass( ... ) )
    {
        /* create the main window */
        flStyle = FCF_STANDARD;
        hwnd = WinCreateStdWindow( ... );
```

```

if( hwnd )
{
    /* Create and associate the help instance */
    hwndHelp = WinCreateHelpInstance( hab, &helpinit );

    if( hwndHelp && WinAssociateHelpInstance( hwndHelp, hwnd ) )
    {
        /* Process messages */
        while( WinGetMsg( hab, &qmsg, NULLHANDLE, 0, 0 ) )
        {
            WinDispatchMsg( hab, &qmsg );
        } /* endwhile */
    }

    /* Remove help instance - note: add */
    /*      WinAssociateHelpInstance( NULLHANDLE, hwnd ); */
    /* to WM_DESTROY processing to remove the association. */
    WinDestroyHelpInstance( hwndHelp );
}

/* finish the cleanup and exit */
WinDestroyMsgQueue( hmq );
WinTerminate( hab );
}

```

WinDestroyHelpInstance - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinDestroyMsgQueue

WinDestroyMsgQueue - Syntax

This function destroys the message queue.

```

#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HMQ    hmq; /* Message-queue handle. */
BOOL    rc; /* Queue-destroyed indicator. */

rc = WinDestroyMsgQueue(hmq);

```

WinDestroyMsgQueue Parameter - hmq

hmq ([HMQ](#)) - input
Message-queue handle.

WinDestroyMsgQueue Return Value - rc

rc ([BOOL](#)) - returns
Queue-destroyed indicator.

TRUE	Queue destroyed
FALSE	Queue not destroyed.

WinDestroyMsgQueue - Parameters

hmq ([HMQ](#)) - input
Message-queue handle.

rc ([BOOL](#)) - returns
Queue-destroyed indicator.

TRUE	Queue destroyed
FALSE	Queue not destroyed.

WinDestroyMsgQueue - Remarks

This function must be called before terminating a thread or an application. Only the thread that called [WinCreateMsgQueue](#) may call this function with that handle.

WinDestroyMsgQueue - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HMQ (0x1002)
An invalid message-queue handle was specified.

WinDestroyMsgQueue - Related Functions

Related Functions

- [WinBroadcastMsg](#)
- [WinCancelShutdown](#)
- [WinCreateMsgQueue](#)
- [WinDestroyMsgQueue](#)
- [WinDispatchMsg](#)
- [WinGetDlgMsg](#)
- [WinGetMsg](#)
- [WinInSendMessage](#)
- [WinPeekMsg](#)
- [WinPostMsg](#)
- [WinPostQueueMsg](#)
- [WinQueryMsgPos](#)
- [WinQueryMsgTime](#)
- [WinQueryQueueInfo](#)
- [WinQueryQueueStatus](#)
- [WinSendDlgItemMsg](#)
- [WinSendMessage](#)
- [WinSetClassMsgInterest](#)
- [WinSetMsgInterest](#)
- [WinSetMsgMode](#)
- [WinSetSynchroMode](#)
- [WinWaitMsg](#)

WinDestroyMsgQueue - Example Code

This example destroys, using WinDestroyMsgQueue, a message queue previously created by WinCreateMsgQueue.

```
#define INCL_WINMESSAGEGR      /* Window Message Functions      */
#define INCL_WINWINDOWMGR      /* Window Manager Functions      */
#include <os2.h>

BOOL      fDestroyed;          /* success of destroy call        */
HAB      hab;                  /* anchor-block handle            */
HMQ      hmq;                  /* message queue handle           */

hab = WinInitialize(0);        /* initialize PM */

hmq = WinCreateMsgQueue(hab, 0); /* create default size queue */

/*
 *   initialize windows, message loop
 */

fDestroyed = WinDestroyMsgQueue(hmq);
```

WinDestroyMsgQueue - Topics

Select an item:

[Syntax](#)

[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinDestroyObject

WinDestroyObject - Syntax

This function is called to delete a workplace object.

```
#define INCL_WINWORKPLACE
#include <os2.h>

HOBJECT    object; /* Handle to a workplace object. */
BOOL       rc;      /* Success indicator. */

rc = WinDestroyObject(object);
```

WinDestroyObject Parameter - object

object ([HOBJECT](#)) - input
Handle to a workplace object.

WinDestroyObject Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

WinDestroyObject - Parameters

object ([HOBJECT](#)) - input
Handle to a workplace object.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

WinDestroyObject - Remarks

The WinDestroyObject function will permanently remove an object that was created with the WinCreateObject function.
Using [HOBJECT](#) for .INI files or files in which an application uses a rename/save/delete sequence is not supported.

WinDestroyObject - Related Functions

Related Functions

- WinCreateObject
- [WinSetObjectData](#)
- WinDestroyObject

WinDestroyObject - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

WinDestroyPointer

WinDestroyPointer - Syntax

This function destroys a pointer or icon.

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HPOINTER    hptrPointer; /* Handle of pointer to be destroyed. */
BOOL        rc;          /* Success indicator. */

rc = WinDestroyPointer(hptrPointer);
```

WinDestroyPointer Parameter - hptrPointer

hptrPointer (**HPOINTER**) - input
Handle of pointer to be destroyed.

WinDestroyPointer Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinDestroyPointer - Parameters

hptrPointer (**HPOINTER**) - input
Handle of pointer to be destroyed.

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinDestroyPointer - Remarks

A pointer can only be destroyed by the thread that created it.

The system pointers and icons must not be destroyed.

WinDestroyPointer - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HPTR (0x101B)
An invalid pointer handle was specified.

WinDestroyPointer - Related Functions

Related Functions

- [WinCreatePointer](#)
 - [WinCreatePointerIndirect](#)
 - [WinDestroyPointer](#)
 - [WinDrawPointer](#)
 - [WinLoadPointer](#)
 - [WinQueryPointer](#)
 - [WinQueryPointerInfo](#)
 - [WinQueryPointerPos](#)
 - [WinQuerySysPointer](#)
 - [WinQuerySysPointerData](#)
 - [WinSetPointer](#)
 - [WinSetPointerPos](#)
 - [WinSetSysPointerData](#)
 - [WinShowPointer](#)
-

WinDestroyPointer - Example Code

This example destroys a bit-map pointer, created by either [WinCreatePointer](#) or [WinCreatePointerIndirect](#), once the window has received a close message (WM_CLOSE).

```
#define INCL_WINPOINTERS          /* Window Pointer Functions */
#define INCL_GPIBITMAPS          /* Graphics Bit-map Functions */
#include <os2.h>
#define IDP_BITMAP 1

HPS    hps;          /* presentation-space handle */
HWND   hwnd;         /* window handle */
HPOINTER hptr;       /* bit-map pointer handle */
HBITMAP hbm;         /* bit-map handle */
BOOL    fSuccess;    /* success indicator */

case WM_CREATE:
    hps = WinBeginPaint(hwnd, NULLHANDLE, NULL);
    hbm = GpiLoadBitmap(hps, 0L, IDP_BITMAP, 64L, 64L);
    WinEndPaint(hps);

    hptr = WinCreatePointer(HWND_DESKTOP, hbm,
                           TRUE, /* use true (system) pointer */
                           0, 0); /* hot spot offset (0,0) */

case WM_CLOSE:
    fSuccess = WinDestroyPointer(hptr);
```

WinDestroyPointer - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinDestroyWindow

WinDestroyWindow - Syntax

This call destroys a window and its child windows.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND    hwnd; /* Window handle. */
BOOL    rc;    /* Window-destroyed indicator. */

rc = WinDestroyWindow(hwnd);
```

WinDestroyWindow Parameter - hwnd

hwnd ([HWND](#)) - input
Window handle.

WinDestroyWindow Return Value - rc

rc ([BOOL](#)) - returns
Window-destroyed indicator.

TRUE	Window destroyed
FALSE	Window not destroyed.

WinDestroyWindow - Parameters

hwnd ([HWND](#)) - input
Window handle.

rc ([BOOL](#)) - returns
Window-destroyed indicator.

TRUE	Window destroyed
FALSE	Window not destroyed.

WinDestroyWindow - Remarks

The window to be destroyed must have been created by the thread that is issuing this call. Before *hwnd* is itself destroyed, all child windows of *hwnd* are also destroyed.

If *hwnd* cannot be destroyed, for example because *hwnd* is an invalid window handle or is not associated with the current thread, *rc* returns FALSE.

Note: If *hwnd* is locked, this call does not return until the window is unlocked (and destroyed).

Messages may be received from other processes or threads during the processing of this call.

If a Presentation Space is associated with the window, it is disassociated from it by this function. If the presentation space was obtained by use of [GpiCreatePS](#) then it is disassociated from the window, but not destroyed. That is, this function calls [GpiAssociate](#) to disassociate the presentation space but does not call [GpiDestroyPS](#). If the presentation space was obtained by use of the [WinGetPS](#) function, it is released by this function; that is, this function performs the [WinReleasePS](#) function.

Messages sent by this call are:

[WM_DESTROY](#)

Always sent to the window being destroyed after the window has been hidden on the device, but before its child windows have been destroyed. The message is sent first to the window being destroyed, then to the child windows as they are destroyed. Therefore, during processing the [WM_DESTROY](#) it can be assumed that all the children still exist.

[WM_ACTIVATE](#)

Sent with if the window being destroyed is the active window.

[WM_RENDERALLFORMATS](#)

Sent if the clipboard owner is being destroyed, and there are unrendered formats in the clipboard.

If the window being destroyed is the active window, both the active window and the input focus window are transferred to another window when the window is destroyed. The window that becomes the active window is the next window, as defined for the "Alt+Esc" function. This usually corresponds to the next application in the sequence. The input focus transfers to whichever window the new active window decides should have it.

If a menu window is being destroyed, any bit maps associated with the menu are not deleted. They will be deleted automatically when the application terminates.

WinDestroyWindow - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinDestroyWindow - Related Functions

Related Functions

- [WinCalcFrameRect](#)
- [WinCreateFrameControls](#)
- [WinCreateStdWindow](#)
- [WinCreateWindow](#)
- [WinDefWindowProc](#)
- [WinDestroyWindow](#)
- [WinQueryClassInfo](#)
- [WinQueryClassName](#)
- [WinRegisterClass](#)
- [WinSubclassWindow](#)

WinDestroyWindow - Related Messages

Related Messages

- [WM_ACTIVATE](#)
- [WM_DESTROY](#)
- [WM_RENDERALLFMTS](#)

WinDestroyWindow - Example Code

This example destroys the specified window and all other windows owned by that window in response to a WM_CLOSE message.

```
#define INCL_WINWINDOWMGR    /* Window Manager functions */
#include <os2.h>

ULONG    fSuccess;
HWND     hwnd;                /* Cursor display window    */

case WM_CLOSE:
    fSuccess = WinDestroyWindow(hwnd);
```

WinDestroyWindow - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinDismissDlg

WinDismissDlg - Syntax

This function hides the modeless dialog window, and causes [WinProcessDlg](#) or [WinDlgBox](#) to return.

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND     hwndDlg; /* Dialog-window handle. */
ULONG    usResult; /* Reply value. */
BOOL     rc;       /* Dialog-dismissed indicator. */

rc = WinDismissDlg(hwndDlg, usResult);
```

WinDismissDlg Parameter - hwndDlg

hwndDlg ([HWND](#)) - input
Dialog-window handle.

WinDismissDlg Parameter - usResult

usResult ([ULONG](#)) - input
Reply value.

Returned to the caller of [WinProcessDlg](#) or [WinDlgBox](#).

WinDismissDlg Return Value - rc

rc ([BOOL](#)) - returns
Dialog-dismissed indicator.

TRUE	Dialog successfully dismissed
FALSE	Dialog not successfully dismissed.

WinDismissDlg - Parameters

hwndDlg ([HWND](#)) - input
Dialog-window handle.

usResult ([ULONG](#)) - input
Reply value.

Returned to the caller of [WinProcessDlg](#) or [WinDlgBox](#).

rc ([BOOL](#)) - returns
Dialog-dismissed indicator.

TRUE	Dialog successfully dismissed
FALSE	Dialog not successfully dismissed.

WinDismissDlg - Remarks

This function is required to complete the processing of a modal dialog window and is called from its dialog procedure. It is made implicitly if the dialog procedure passes a [WM_COMMAND](#) message to [WinDefDlgProc](#) or if a [WM_QUIT](#) message is encountered during [WinProcessDlg](#) or [WinGetDlgMsg](#).

This function hides the dialog window, and re-enables any windows that were disabled by [WinProcessDlg](#) or [WinGetDlgMsg](#).

If the dialog window needs to be processed through [WinProcessDlg](#), after being dismissed, the [FF_DLGDISMISSED](#) flag must be reset.

It does not destroy the dialog window; [WinDestroyWindow](#) must be issued to destroy the dialog window when it is no longer needed. However, [WinDlgBox](#) destroys the dialog window it creates, when the dialog window is dismissed by the use of this function.

This function can be issued during the processing of the the [WM_INITDLG \(Default Dialogs\)](#) message.

Note: This function can be made from a modeless dialog window, although this is not necessary as there is no internal message processing loop. If it is called, the dialog window is hidden and it is the responsibility of the application to destroy the dialog window, if required.

WinDismissDlg - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)

An invalid window handle was specified.

WinDismissDlg - Related Functions

Related Functions

- [WinCreateDlg](#)
 - [WinDefDlgProc](#)
 - [WinDismissDlg](#)
 - [WinDlgBox](#)
 - [WinGetDlgMsg](#)
 - [WinLoadDlg](#)
 - [WinProcessDlg](#)
-

WinDismissDlg - Related Messages

Related Messages

- [WM_COMMAND](#)
 - [WM_QUIT](#)
 - [WM_INITDLG](#) (Default Dialogs)
-

WinDismissDlg - Example Code

This example shows a typical dialog procedure that has both an OK and a Cancel button. If the user selects the OK button, WinDismissDlg is called with a result value of TRUE. If the user selects the Cancel button, WinDismissDlg is called with a result value of FALSE.

```
#define INCL_WINDIALOGS    /* Window Dialog Mgr Functions */
#define ID_ENTER  101;
#define ID_CANCEL 102;
#include <os2.h>

MPARAM  mpParam1;
HWND    hwnd;

case WM_COMMAND:
    switch (SHORT1FROMMP(mpParam1))
    {
        /* OK button selected */
        case ID_ENTER:
            WinDismissDlg(hwnd, TRUE);
            return (0L);

        /* Cancel button selected */
        case ID_CANCEL:
            WinDismissDlg(hwnd, FALSE);
            return (0L);
    }
}
```

WinDismissDlg - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinDispatchMsg

WinDispatchMsg - Syntax

This function invokes a window procedure.

```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
PQMSG    pqmsgMsg;     /* Message structure. */
MRESULT  mresReply;    /* Message-return data. */

mresReply = WinDispatchMsg(hab, pqmsgMsg);
```

WinDispatchMsg Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinDispatchMsg Parameter - pqmsgMsg

pqmsgMsg ([PQMSG](#)) - input
Message structure.

WinDispatchMsg Return Value - mresReply

mresReply ([MRESULT](#)) - returns
Message-return data.

WinDispatchMsg - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

pqmsgMsg ([PQMSG](#)) - input
Message structure.

mresReply ([MRESULT](#)) - returns
Message-return data.

WinDispatchMsg - Remarks

This function is equivalent to using the [WinSendMsg](#) function with the parameters corresponding to those in *pqmsgMsg*.

The time and pointer position information within *pqmsgMsg* can be obtained by the window procedure with the [WinQueryMsgTime](#) and [WinQueryMsgPos](#) functions.

mresReply is the value returned by the invoked window procedure. For standard window classes, the values of *mresReply* are documented with the message definitions;

WinDispatchMsg - Related Functions

Related Functions

- [WinBroadcastMsg](#)
- [WinCancelShutdown](#)
- [WinCreateMsgQueue](#)
- [WinDestroyMsgQueue](#)
- [WinDispatchMsg](#)
- [WinGetDlgMsg](#)
- [WinGetMsg](#)
- [WinInSendMsg](#)
- [WinPeekMsg](#)
- [WinPostMsg](#)
- [WinPostQueueMsg](#)
- [WinQueryMsgPos](#)
- [WinQueryMsgTime](#)
- [WinQueryQueueInfo](#)
- [WinQueryQueueStatus](#)
- [WinSendDlgItemMsg](#)
- [WinSendMsg](#)
- [WinSetClassMsgInterest](#)
- [WinSetMsgInterest](#)
- [WinSetMsgMode](#)
- [WinSetSynchroMode](#)
- [WinWaitMsg](#)

WinDispatchMsg - Example Code

This example, after uses WinDispatchMsg within a WinGetMsg loop to dispatch window messages to a window procedure.

```
#define INCL_WINMESSAGEGR      /* Window Message Functions */
#define INCL_WINWINDOWMGR     /* Window Manager Functions */
#include <os2.h>

HAB      hab;                /* anchor-block handle */
HMQ      hmq;                /* message queue handle */
QMSG     qmsg;               /* message */

hab = WinInitialize(0);      /* initialize PM */

hmq = WinCreateMsgQueue(hab, 0); /* create default size queue */

/*
.
.  initialize windows
.
*/

/* get and dispatch messages from queue */
while (WinGetMsg(hab, &qmsg, 0, 0, 0))
    WinDispatchMsg(hab, &qmsg);
```

WinDispatchMsg - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinDlgBox

WinDlgBox - Syntax

This function loads and processes a modal dialog window and returns the result value established by the [WinDismissDlg](#) call.

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND      hwndParent;        /* Parent-window handle of the created dialog window. */
HWND      hwndOwner;         /* Requested owner-window handle of the created dialog window. */
```

```

PFNWP      pfnDlgProc;      /* Dialog procedure for the created dialog window. */
HMODULE     hmod;           /* Resource identity containing the dialog template. */
ULONG      idDlg;          /* Dialog-template identity within the resource file. */
PVOID      pCreateParams;  /* Pointer to application-defined data area. */
ULONG      ulResult;       /* Reply value. */

ulResult = WinDlgBox(hwndParent, hwndOwner,
                    pfnDlgProc, hmod, idDlg, pCreateParams);

```

WinDlgBox Parameter - hwndParent

hwndParent ([HWND](#)) - input

Parent-window handle of the created dialog window.

HWND_DESKTOP

The desktop window

HWND_OBJECT

Object window

Other

Specified window.

WinDlgBox Parameter - hwndOwner

hwndOwner ([HWND](#)) - input

Requested owner-window handle of the created dialog window.

The actual owner window is calculated using the algorithm specified in the description of the [WinLoadDlg](#) function.

WinDlgBox Parameter - pfnDlgProc

pfnDlgProc ([PFNWP](#)) - input

Dialog procedure for the created dialog window.

WinDlgBox Parameter - hmod

hmod ([HMODULE](#)) - input

Resource identity containing the dialog template.

NULLHANDLE

Use the application's .EXE file.

Other

Module handle returned from the DosLoadModule or DosQueryModuleHandle call.

WinDlgBox Parameter - idDlg

idDlg ([ULONG](#)) - input

Dialog-template identity within the resource file.

It is also used as the identity of the created dialog window. Is must be greater or equal to 0 and less or equal to 0xFFFF.

WinDlgBox Parameter - pCreateParams

pCreateParams ([PVOID](#)) - input

Pointer to application-defined data area.

This is passed to the dialog procedure in the [WM_INITDLG](#) message.

This parameter MUST be a pointer rather than a long.

WinDlgBox Return Value - ulResult

ulResult ([ULONG](#)) - returns

Reply value.

Value established by the [WinDismissDlg](#) call or DID_ERROR if an error occurs.

WinDlgBox - Parameters

hwndParent ([HWND](#)) - input

Parent-window handle of the created dialog window.

HWND_DESKTOP

The desktop window

HWND_OBJECT

Object window

Other

Specified window.

hwndOwner ([HWND](#)) - input

Requested owner-window handle of the created dialog window.

The actual owner window is calculated using the algorithm specified in the description of the [WinLoadDlg](#) function.

pfnDlgProc ([PFNWP](#)) - input
Dialog procedure for the created dialog window.

hmod ([HMODULE](#)) - input
Resource identity containing the dialog template.

NULLHANDLE
Use the application's .EXE file.
Other
Module handle returned from the [DosLoadModule](#) or [DosQueryModuleHandle](#) call.

idDlg ([ULONG](#)) - input
Dialog-template identity within the resource file.

It is also used as the identity of the created dialog window. Its must be greater or equal to 0 and less or equal to 0xFFFF.

pCreateParams ([PVOID](#)) - input
Pointer to application-defined data area.

This is passed to the dialog procedure in the [WM_INITDLG](#) message.

This parameter MUST be a pointer rather than a long.

ulResult ([ULONG](#)) - returns
Reply value.

Value established by the [WinDismissDlg](#) call or [DID_ERROR](#) if an error occurs.

WinDlgBox - Remarks

The use of parameters to this function are the same as those of the [WinLoadDlg](#) function.

This function should not be used while pointing device capture is set (see [WinSetCapture](#)).

This function does not return until [WinDismissDlg](#) is called.

This function is equivalent to:

```
WinLoadDlg (., ., ., ., ., ., ., dlg);  
WinProcessDlg (dlg, result);  
WinDestroyWindow (dlg, success);  
return (result);
```

and the remarks documented under these calls also apply.

If a dialog template (typically compiled using the resource compiler) references another resource (for example an icon resource for an icon static control), this function always searches for that resource in the .EXE file. If an application wishes to keep resources referenced by a dialog template in a .DLL library, these resources must be loaded by an explicit function call during the processing of the [WM_INITDLG](#) message.

This can be considered to be a customizable "read from screen" call. The caller supplies a data buffer (the *pCreateParams* parameter), filled with initial values. It receives a return code which indicates whether the data in the buffer has been updated and validated, or whether the end user cancelled the dialog. The end user interface is encapsulated within the dialog window. The dialog template provides a view of the current state of the data buffer, the dialog procedure defines how the user can change the data. The caller need know nothing about the details of the end user interface. It makes a single "read from screen" call and continues with its work.

Note: If a dialog box or a message box is up for a window, and the parent or owner of that window is destroyed, the code following the [WinDlgBox](#) or [WinMessageBox](#) call is executed even though the parent/owner window no longer exists. This can result in accessing data that no longer exists; especially data referenced in the window words. Therefore, it is extremely important to determine the state of your child-window procedure after this function returns. The most straightforward method for doing this is to call [WinQueryWindowPtr](#) to get a pointer to the window words. If the returned pointer is NULL, then you should exit immediately. Should this be the case, the bottom-up rule (that is, the child window gets [WM_DESTROY](#) messages first, then the parent window) still applies, and it becomes the child window procedure's responsibility to exit gracefully.

WinDlgBox - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)

An invalid window handle was specified.

PMERR_INVALID_INTEGER_ATOM (0x1016)

The specified atom is not a valid integer atom.

PMERR_INVALID_ATOM_NAME (0x1015)

An invalid atom name string was passed.

PMERR_ATOM_NAME_NOT_FOUND (0x1017)

The specified atom name is not in the atom table.

PMERR_RESOURCE_NOT_FOUND (0x100A)

The specified resource identity could not be found.

WinDlgBox - Related Functions

Related Functions

- [WinCreateDlg](#)
 - [WinDefDlgProc](#)
 - [WinDismissDlg](#)
 - [WinDlgBox](#)
 - [WinGetDlgMsg](#)
 - [WinLoadDlg](#)
 - [WinProcessDlg](#)
-

WinDlgBox - Related Messages

Related Messages

- [WM_INITDLG](#)
-

WinDlgBox - Example Code

This example processes an application-defined message (IDM_OPEN) and calls WinDlgBox to load a dialog box.

```
#define IDD_OPEN 1
#define INCL_WINDIALOGS          /* Window Dialog Mgr Functions */
#include <os2.h>

HWND    hwndFrame;                /* frame window handle */
PFNWP   OpenDlg;

case IDM_OPEN:
```

```

if (WinDlgBox(HWND_DESKTOP,
    hwndFrame,      /* handle of the owner      */
    OpenDlg,        /* dialog procedure address */
    NULLHANDLE,     /* location of dialog resource */
    IDD_OPEN,       /* resource identifier      */
    NULL)) {        /* application-specific data */
    .
. /* code executed if dialog box returns TRUE */
.
}

```

WinDlgBox - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinDrawBitmap

WinDrawBitmap - Syntax

This function draws a bit map using the current image colors and mixes.

```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HPS      hpsDst; /* Handle of presentation space in which the bit map is drawn. */
HBITMAP  hbm;    /* Bit-map handle. */
PRECTL   pwrcSrc; /* Subrectangle of bit map to be drawn. */
PPOINTL  pptlDst; /* Bit-map destination. */
LONG     clrFore; /* Foreground color. */
LONG     clrBack; /* Background color. */
ULONG    fl;      /* Flags that determine how the bit map is drawn. */
BOOL     rc;      /* Success indicator. */

rc = WinDrawBitmap(hpsDst, hbm, pwrcSrc, pptlDst,
    clrFore, clrBack, fl);

```

WinDrawBitmap Parameter - hpsDst

hpsDst ([HPS](#)) - input
Handle of presentation space in which the bit map is drawn.

WinDrawBitmap Parameter - hbm

hbm ([HBITMAP](#)) - input
Bit-map handle.

WinDrawBitmap Parameter - pwrcSrc

pwrcSrc ([PRECTL](#)) - input
Subrectangle of bit map to be drawn.

Note: The value of each field in this structure must be in the range -32 768 through 32 768. The data type [WRECT](#) can also be used, if supported by the language.

- | | |
|-------|--|
| NULL | The whole of the bit map is drawn |
| Other | The whole of the bit map is not drawn. |

WinDrawBitmap Parameter - pptIDst

pptIDst ([PPOINTL](#)) - input
Bit-map destination.

The bottom left corner of the bit-map destination is specified in device coordinates.

If DBM_STRETCH is set in *hbm*, this parameter is used as a pointer to a [RECTL](#) data structure that contains the coordinates of the rectangle into which the source bit map is to be drawn. In this situation, *pptIDst* should be treated as a PRECTL datatype.

WinDrawBitmap Parameter - clrFore

clrFore ([LONG](#)) - input
Foreground color.

This is used if *hbm* refers to a monochrome bit map. In this instance, bit-map bits that are set to 1 are drawn using *clrFore*. Ignored if DBM_IMAGEATTRS is specified.

WinDrawBitmap Parameter - clrBack

clrBack (**LONG**) - input
Background color.

This is used if *hbm* refers to a monochrome bit map. In this instance, bit-map bits that are set to zero are drawn using *clrBack*. Ignored if DBM_IMAGEATTRS is specified.

WinDrawBitmap Parameter - fl

fl (**ULONG**) - input
Flags that determine how the bit map is drawn.

- DBM_NORMAL Draw the bit map normally using ROP_SRCCOPY, as defined in GpiBitBlt.
- DBM_INVERT Draw the bit map inverted using ROP_NOTSRCCOPY, as defined in GpiBitBlt.
- DBM_STRETCH *pptlDst* is used to point to a **RECTL** data structure representing a rectangle in the destination presentation space, into which the bit map will be stretched or compressed. If compression is required, some rows and columns of the bit map are eliminated.
- DBM_HALFTONE Use the OR operator to combine the bit map with an alternating pattern of ones or zeros before drawing it. It can be used with either DBM_NORMAL or DBM_INVERT.
- DBM_IMAGEATTRS If this is specified, color conversion of monochrome bit maps is done by using the image attributes.

WinDrawBitmap Return Value - rc

rc (**BOOL**) - returns
Success indicator.

- TRUE Successful completion
- FALSE Error occurred.

WinDrawBitmap - Parameters

hpsDst (**HPS**) - input
Handle of presentation space in which the bit map is drawn.

hbm ([HBITMAP](#)) - input
Bit-map handle.

pwrcSrc ([PRECTL](#)) - input
Subrectangle of bit map to be drawn.

Note: The value of each field in this structure must be in the range -32 768 through 32 768.
The data type [WRECT](#) can also be used, if supported by the language.

NULL
The whole of the bit map is drawn

Other
The whole of the bit map is not drawn.

pptlDst ([PPOINTL](#)) - input
Bit-map destination.

The bottom left corner of the bit-map destination is specified in device coordinates.

If DBM_STRETCH is set in *fl*, this parameter is used as a pointer to a [RECTL](#) data structure that contains the coordinates of the rectangle into which the source bit map is to be drawn. In this situation, *pptlDst* should be treated as a PRECTL datatype.

clrFore ([LONG](#)) - input
Foreground color.

This is used if *hbm* refers to a monochrome bit map. In this instance, bit-map bits that are set to 1 are drawn using *clrFore*. Ignored if DBM_IMAGEATTRS is specified.

clrBack ([LONG](#)) - input
Background color.

This is used if *hbm* refers to a monochrome bit map. In this instance, bit-map bits that are set to zero are drawn using *clrBack*. Ignored if DBM_IMAGEATTRS is specified.

fl ([ULONG](#)) - input
Flags that determine how the bit map is drawn.

DBM_NORMAL
Draw the bit map normally using ROP_SRCCOPY, as defined in GpiBitBlt.

DBM_INVERT
Draw the bit map inverted using ROP_NOTSRCCOPY, as defined in GpiBitBlt.

DBM_STRETCH
pptlDst is used to point to a [RECTL](#) data structure representing a rectangle in the destination presentation space, into which the bit map will be stretched or compressed. If compression is required, some rows and columns of the bit map are eliminated.

DBM_HALFTONE
Use the OR operator to combine the bit map with an alternating pattern of ones or zeros before drawing it. It can be used with either DBM_NORMAL or DBM_INVERT.

DBM_IMAGEATTRS
If this is specified, color conversion of monochrome bit maps is done by using the image attributes.

rc ([BOOL](#)) - returns
Success indicator.

TRUE
Successful completion

FALSE
Error occurred.

WinDrawBitmap - Remarks

This function should only be used in draw mode (DM_DRAW) to a screen device context (see "GpiSetDrawingMode" in the *Graphics*

Programming Interface Programming Reference). The presentation space handle can be to either a micro-presentation space or a normal presentation space (see "GpiCreatePS" in the *Graphics Programming Interface Programming Reference*).

If *hbm* refers to a color bit map, no color conversion is performed.

The current position in the presentation space is not changed by this function.

WinDrawBitmap - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_FLAG (0x1019)

An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

PMERR_HBITMAP_BUSY (0x2032)

An internal bit map busy error was detected. The bit map was locked by one thread during an attempt to access it from another thread.

WinDrawBitmap - Related Functions

Related Functions

- [WinDrawBitmap](#)
- [WinDrawBorder](#)
- [WinDrawPointer](#)
- [WinDrawText](#)
- [WinFillRect](#)
- [WinGetSysBitmap](#)
- [WinInvertRect](#)
- [WinQueryPresParam](#)
- [WinRemovePresParam](#)
- [WinScrollWindow](#)
- [WinSetPresParam](#)

WinDrawBitmap - Example Code

This example uses WinDrawBitmap to draw the system-defined menu check mark bit map in response to the user selecting a menu item (WM_MENUSELECT), using the bit-map handle returned by WinGetSysBitmap.

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#define INCL_WINPOINTERS      /* Window Pointer Functions */
#define INCL_WINMESSAGEGR     /* Window Message Functions */
#define INCL_WINMENUS         /* Window Menu Functions */

#include <os2.h>

HPS    hps;          /* presentation-space handle */
HBITMAP hbmCheck;    /* check mark bit-map handle */
HWND   hwndMenu;     /* menu handle */
USHORT usItemId;     /* menu item id */
RECT   rcItem;       /* item border rectangle */
MPARAM mpParam1;     /* Parameter 1 (menu item id) */
MPARAM mpParam2;     /* Parameter 2 (menu handle) */

case WM_CREATE:
    /* obtain check mark bit-map handle */
```

```

        hbmCheck = WinGetSysBitmap(HWND_DESKTOP, SBMP_MENUCHECK);

case WM_MENUSELECT:
    usItemId = SHORT1FROMMP(mpParam1);
    hwndMenu = HWNDFROMMP(mpParam2);

    /* get rectangle of selected item */
    WinSendMsg(hwndMenu,
        MM_QUERYITEMRECT,
        MPFROM2SHORT(usItemId, TRUE),
        MPFROMP(&rclItem));

    /* draw the check mark in the lower left corner of item's
       rectangle */
    if (hbmCheck != NULL)
    {
        WinDrawBitmap(hps,
            hbmCheck,          /* check mark          */
            NULL,             /* draw whole bit map  */
            (PPOINTL)&rclItem, /* bit-map destination */
            0L,               /* ignored since color  */
            0L,               /* bit map             */
            DBM_NORMAL);      /* draw normal size     */
    }
}

```

WinDrawBitmap - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinDrawBorder

WinDrawBorder - Syntax

This function draws the borders and interior of a rectangle.

```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HPS      hps;          /* Presentation-space handle. */
PRECTL   prcl;         /* Bounding rectangle for the border. */
LONG     cx;           /* Width of border rectangle vertical sides. */
LONG     cy;           /* Width of border rectangle horizontal sides. */
LONG     clrFore;      /* Color of edge of border. */
LONG     clrBack;      /* Color of interior of border. */
ULONG    flCmd;        /* Flags controlling the way in which the border is drawn. */
BOOL     rc;           /* Success indicator. */

```

```
rc = WinDrawBorder(hps, prcl, cx, cy, clrFore,  
                  clrBack, flCmd);
```

WinDrawBorder Parameter - hps

hps ([HPS](#)) - input
Presentation-space handle.

WinDrawBorder Parameter - prcl

prcl ([PRECTL](#)) - input
Bounding rectangle for the border.

The rectangle is in device coordinates.

The border is drawn within the rectangle. Along the bottom and left edges of the rectangle, the edges of the border coincide with the rectangle edges. Along the top and right edges of the rectangle, the border is drawn one device unit inside the rectangle edges.

Note: The value of each field in this structure must be in the range -32 768 through 32 768. The data type [WRECT](#) can also be used, if supported by the language.

WinDrawBorder Parameter - cx

cx ([LONG](#)) - input
Width of border rectangle vertical sides.

This is the width of the left and right sides in device coordinates.

WinDrawBorder Parameter - cy

cy ([LONG](#)) - input
Width of border rectangle horizontal sides.

This is the width of the top and bottom sides in device coordinates.

WinDrawBorder Parameter - clrFore

clrFore ([LONG](#)) - input
Color of edge of border.

Not used if DB_AREATTRS is specified.

WinDrawBorder Parameter - clrBack

clrBack ([LONG](#)) - input
Color of interior of border.

Not used if DB_AREATTRS is specified.

WinDrawBorder Parameter - flCmd

flCmd ([ULONG](#)) - input
Flags controlling the way in which the border is drawn.

Some of the DB_* flags are mutually exclusive. Only one of these four can be significant:

- DB_PATCOPY (default)
- DB_PATINVERT
- DB_DESTINVERT
- DB_AREAMIXMODE.

Possible values are described in the following list:

DB_ROP	A group of flags that specify the mix to be used, for both the border and the interior.
DB_PATCOPY	Use the ROP_PATCOPY raster operation (see "GpiBitBlt" in the <i>Graphics Programming Interface Programming Reference</i>). This is a copy of the pattern to the destination.
DB_PATINVERT	Use the ROP_PATINVERT raster operation (see "GpiBitBlt" in the <i>Graphics Programming Interface Programming Reference</i>). This is an exclusive-OR of the pattern with the destination.
DB_DESTINVERT	Use the ROP_DESTINVERT raster operation (see "GpiBitBlt" in the <i>Graphics Programming Interface Programming Reference</i>). This inverts the destination.
DB_AREAMIXMODE	Map the current area foreground mix attribute into a BitBlt raster operation (see "GpiBitBlt" in the <i>Graphics Programming Interface Programming Reference</i>). The area background mix mode is ignored.
DB_INTERIOR	The area contained within the given rectangle, and not included within the borders (as given by <i>cx</i> and <i>cy</i>), is drawn.
DB_AREATTRS	

- If this is specified:

For any border, the pattern used is the pattern as currently defined in the area attribute.

For any interior, the pattern used is the same as if GpiSetAttrs for the area attributes is made with the background color of the area attribute being passed for the foreground color, and the foreground color of the area attribute being passed as the background color.

- If this is not specified (default):

For any border, the pattern used is the same as if GpiSetAttrs for the area attributes is made with a foreground color of *clrFore*, and a background color of *clrBack*.

For any interior, the pattern used is the same as if GpiSetAttrs for the area attributes is made with a foreground color of *clrBack*, and a background color of *clrFore*.

DB_STANDARD

cx and *cy* are multiplied by the system SV_CXBORDER and SV_CYBORDER constants to produce the widths of the vertical and horizontal sides of the border.

DB_DLGBOARDER

A standard dialog border is drawn, in the active titlebar color if DB_PATCOPY is specified, or the inactive titlebar color if DB_PATINVERT is specified. Other DB_ROP options, and DB_AREAATTRS, are ignored.

DB_ROP and DB_AREAATTRS are also ignored for the interior. The interior is drawn in the color specified by *clrBack*.

WinDrawBorder Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinDrawBorder - Parameters

hps (**HPS**) - input
Presentation-space handle.

prcl (**PRECTL**) - input
Bounding rectangle for the border.

The rectangle is in device coordinates.

The border is drawn within the rectangle. Along the bottom and left edges of the rectangle, the edges of the border coincide with the rectangle edges. Along the top and right edges of the rectangle, the border is drawn one device unit inside the rectangle edges.

Note: The value of each field in this structure must be in the range -32 768 through 32 768. The data type **WRECT** can also be used, if supported by the language.

cx (**LONG**) - input
Width of border rectangle vertical sides.

This is the width of the left and right sides in device coordinates.

cy (**LONG**) - input
Width of border rectangle horizontal sides.

This is the width of the top and bottom sides in device coordinates.

clrFore ([LONG](#)) - input
Color of edge of border.

Not used if DB_AREAATTRS is specified.

clrBack ([LONG](#)) - input
Color of interior of border.

Not used if DB_AREAATTRS is specified.

flCmd ([ULONG](#)) - input
Flags controlling the way in which the border is drawn.

Some of the DB_* flags are mutually exclusive. Only one of these four can be significant:

- DB_PATCOPY (default)
- DB_PATINVERT
- DB_DESTINVERT
- DB_AREAMIXMODE.

Possible values are described in the following list:

DB_ROP
A group of flags that specify the mix to be used, for both the border and the interior.

DB_PATCOPY
Use the ROP_PATCOPY raster operation (see "GpiBitBlt" in the *Graphics Programming Interface Programming Reference*). This is a copy of the pattern to the destination.

DB_PATINVERT
Use the ROP_PATINVERT raster operation (see "GpiBitBlt" in the *Graphics Programming Interface Programming Reference*). This is an exclusive-OR of the pattern with the destination.

DB_DESTINVERT
Use the ROP_DESTINVERT raster operation (see "GpiBitBlt" in the *Graphics Programming Interface Programming Reference*). This inverts the destination.

DB_AREAMIXMODE
Map the current area foreground mix attribute into a Bitblt raster operation (see "GpiBitBlt" in the *Graphics Programming Interface Programming Reference*). The area background mix mode is ignored.

DB_INTERIOR
The area contained within the given rectangle, and not included within the borders (as given by *cx* and *cy*), is drawn.

DB_AREAATTRS

- If this is specified:

For any border, the pattern used is the pattern as currently defined in the area attribute.

For any interior, the pattern used is the same as if GpiSetAttrs for the area attributes is made with the background color of the area attribute being passed for the foreground color, and the foreground color of the area attribute being passed as the background color.
- If this is not specified (default):

For any border, the pattern used is the same as if GpiSetAttrs for the area attributes is made with a foreground color of *clrFore*, and a background color of *clrBack*.

For any interior, the pattern used is the same as if GpiSetAttrs for the area attributes is made with a foreground color of *clrBack*, and a background color of *clrFore*.

DB_STANDARD
cx and *cy* are multiplied by the system SV_CXBORDER and SV_CYBORDER constants to produce the widths of the vertical and horizontal sides of the border.

DB_DLGBORDER
A standard dialog border is drawn, in the active titlebar color if DB_PATCOPY is specified, or the inactive titlebar color if DB_PATINVERT is specified. Other DB_ROP options, and DB_AREAATTRS, are ignored.

DB_ROP and DB_AREATTRS are also ignored for the interior. The interior is drawn in the color specified by *clrBack*.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinDrawBorder - Remarks

A border is a rectangular frame, normally used around the edge of a window.

This function should only be used in draw mode (DM_DRAW), to a screen device context; *hps* can be either a micro-presentation space or a normal presentation space (see "GpiCreatePS" in the *Graphics Programming Interface Programming Reference*). DB_DESTINVERT inverts the destination.

If DB_AREAMIXMODE is given, the foreground mix mode from the area attribute is mapped into an equivalent ROP_ value (see "GpiBitBlt" in the *Graphics Programming Interface Programming Reference*). The area background mix mode is ignored.

Either or both *cx* or *cy* can be zero. If both are zero, the interior is still drawn. If either the x borders overlap or the y borders overlap, the border is drawn as a single rectangle with no interior.

WinDrawBorder - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_FLAG (0x1019)

An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

PMERR_INV_DRAW_BORDER_OPTION (0x2068)

An invalid option parameter was specified with WinDrawBorder.

WinDrawBorder - Related Functions

Related Functions

- [WinDrawBitmap](#)
- [WinDrawBorder](#)
- [WinDrawPointer](#)
- [WinDrawText](#)
- [WinFillRect](#)
- [WinGetSysBitmap](#)
- [WinInvertRect](#)
- [WinQueryPresParam](#)
- [WinRemovePresParam](#)
- [WinScrollWindow](#)
- [WinSetPresParam](#)

WinDrawBorder - Example Code

This example uses WinDrawBorder to draw the border (width of 5) and interior of a 300x200 rectangle anchored at (0,0), and using the area's current attributes for both the border and interior colors.

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#include <os2.h>

HPS    hps;          /* presentation-space handle */
BOOL    fSuccess;     /* success indicator */
RECTL   prclRectangle={0,0,300,200}; /* border rectangle */
LONG    lVertSideWidth=5; /* Width of border rectangle vertical
                           sides */
LONG    lHorizSideWidth=5; /* Width of border rectangle horizontal
                           sides */
ULONG    flCmd;       /* draw flags */

/* use current area attributes */
flCmd = DB_AREAATTRS;

fSuccess = WinDrawBorder(hps, &prclRectangle, lVertSideWidth,
                        lHorizSideWidth, 0L, 0L, flCmd);
```

WinDrawBorder - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinDrawPointer

WinDrawPointer - Syntax

This function draws a pointer in the passed *hps* at the passed coordinates [*lx*, *ly*].

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HPS    hps;          /* Presentation-space handle into which the pointer is drawn. */
LONG    lx;          /* x-coordinate at which to draw the pointer, in device coordinates. */
LONG    ly;          /* y-coordinate at which to draw the pointer, in device coordinates. */
HPOINTER hpPtrPointer; /* Pointer handle. */
ULONG    ulHalftone;  /* Shading control with which to draw the pointer. */
BOOL    rc;          /* Success indicator. */

rc = WinDrawPointer(hps, lx, ly, hpPtrPointer,
                    ulHalftone);
```

WinDrawPointer Parameter - hps

hps ([HPS](#)) - input

Presentation-space handle into which the pointer is drawn.

This can be either a micro presentation space or a normal presentation space (see "GpiCreatePS" in the *Graphics Programming Interface Programming Reference*).

WinDrawPointer Parameter - lx

lx ([LONG](#)) - input

x-coordinate at which to draw the pointer, in device coordinates.

WinDrawPointer Parameter - ly

ly ([LONG](#)) - input

y-coordinate at which to draw the pointer, in device coordinates.

WinDrawPointer Parameter - hptrPointer

hptrPointer ([HPOINTER](#)) - input

Pointer handle.

The pointer should be loaded using [WinLoadPointer](#), [WinCreatePointer](#), or [WinCreatePointerIndirect](#).

WinDrawPointer Parameter - ulHalftone

ulHalftone ([ULONG](#)) - input

Shading control with which to draw the pointer.

DP_NORMAL

As it normally appears.

DP_HALFTONED

With a halftone pattern where black normally appears.

DP_INVERTED	Inverted, black for white and white for black.
DP_MINIICON	Bit map of a mini icon.

WinDrawPointer Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Function failed.

WinDrawPointer - Parameters

hps (**HPS**) - input
Presentation-space handle into which the pointer is drawn.

This can be either a micro presentation space or a normal presentation space (see "GpiCreatePS" in the *Graphics Programming Interface Programming Reference*).

lx (**LONG**) - input
x-coordinate at which to draw the pointer, in device coordinates.

ly (**LONG**) - input
y-coordinate at which to draw the pointer, in device coordinates.

hptrPointer (**HPOINTER**) - input
Pointer handle.

The pointer should be loaded using [WinLoadPointer](#), [WinCreatePointer](#), or [WinCreatePointerIndirect](#).

ulHalftone (**ULONG**) - input
Shading control with which to draw the pointer.

DP_NORMAL	As it normally appears.
DP_HALFTONED	With a halftone pattern where black normally appears.
DP_INVERTED	Inverted, black for white and white for black.
DP_MINIICON	Bit map of a mini icon.

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Function failed.

WinDrawPointer - Remarks

This function should only be used in draw mode (DM_DRAW) to a screen device context.

WinDrawPointer - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HPTR (0x101B)
An invalid pointer handle was specified.

PMERR_INVALID_FLAG (0x1019)
An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

WinDrawPointer - Related Functions

Related Functions

- [WinCreatePointer](#)
 - [WinCreatePointerIndirect](#)
 - [WinDestroyPointer](#)
 - [WinDrawBitmap](#)
 - [WinDrawBorder](#)
 - [WinDrawPointer](#)
 - [WinDrawText](#)
 - [WinFillRect](#)
 - [WinGetSysBitmap](#)
 - [WinInvertRect](#)
 - [WinLoadPointer](#)
 - [WinQueryPointer](#)
 - [WinQueryPointerInfo](#)
 - [WinQueryPointerPos](#)
 - [WinQueryPresParam](#)
 - [WinQuerySysPointer](#)
 - [WinQuerySysPointerData](#)
 - [WinRemovePresParam](#)
 - [WinScrollWindow](#)
 - [WinSetPointer](#)
 - [WinSetPointerPos](#)
 - [WinSetPresParam](#)
 - [WinShowPointer](#)
 - [WinDrawPointer](#)
 - [WinSetSysPointerData](#)
-

WinDrawPointer - Example Code

This example draws a bit map pointer, created by either WinCreatePointer or WinCreatePointerIndirect, in response to a paint message (WM_PAINT).

```
#define INCL_WINPOINTERS      /* Window Pointer Functions */
#define INCL_GPIBITMAPS      /* Graphics bit-map functions */
#include <os2.h>

HPS    hps;          /* presentation-space handle */
HWND   hwnd;         /* window handle */
```

```

HPOINTER  hptr;           /* bit-map pointer handle          */
HBITMAP   hbm;           /* bit-map handle          */
BOOL      fSuccess;       /* success indicator       */
ULONG     ulHalftone=DP_NORMAL; /* draw with normal shading */

case WM_CREATE:
    hps = WinBeginPaint(hwnd, NULLHANDLE, NULL);
    hbm = GpiLoadBitmap(hps, 0L, IDP_BITMAP, 64L, 64L);
    WinEndPaint(hps);

    hptr = WinCreatePointer(HWND_DESKTOP, hbm,
                           TRUE, /* use true (system) pointer */
                           0, 0); /* hot spot offset (0,0) */

case WM_PAINT:
    hps = WinBeginPaint(hwnd, NULLHANDLE, NULL);
    fSuccess = WinDrawPointer(hps, 50, 50, hptr, ulHalftone);
    WinEndPaint(hps);

```

WinDrawPointer - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinDrawText

WinDrawText - Syntax

This function draws a single line of formatted text into a specified rectangle.

```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HPS      hps;           /* Presentation-space handle. */
LONG     cchText;       /* Count of the number of characters in the string. */
PCH      lpchText;     /* Character string to be drawn. */
PRECTL   prcl;         /* Text rectangle. */
LONG     clrFore;       /* Foreground color. */
LONG     clrBack;       /* Background color. */
ULONG     flCmd;        /* An array of flags that determines how the text is drawn. */
LONG     lChars;        /* Count of characters drawn within the rectangle. */

lChars = WinDrawText(hps, cchText, lpchText,
                    prcl, clrFore, clrBack, flCmd);

```

WinDrawText Parameter - hps

hps ([HPS](#)) - input
Presentation-space handle.

WinDrawText Parameter - cchText

cchText ([LONG](#)) - input
Count of the number of characters in the string.

This parameter must be greater or equal to -1L.

-1L The string is null-terminated and its length is to be calculated by this function.

Other Count of the number of characters in the string.

WinDrawText Parameter - lpchText

lpchText ([PCH](#)) - input
Character string to be drawn.

A carriage-return or line-feed character always terminates a line. If the length of a line is less than *cchText* (the string is divided into more than one line) each line is terminated by either of the mentioned characters.

WinDrawText Parameter - prcl

prcl ([PRECTL](#)) - in/out
Text rectangle.

Rectangle within which the text is to be formatted, in world coordinates. Points on the boundary of this rectangle are deemed to be inside the rectangle.

On input, this rectangle is the desired rectangle in which the text is to be drawn.

On output, the height of the rectangle is modified to the actual size needed to draw the given text string. The return value is only of interest in the instance where DT_QUERYEXTENT is set in *flCmd*.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

WinDrawText Parameter - clrFore

clrFore ([LONG](#)) - input
Foreground color.

Ignored if DT_TEXTATTRS is specified.

WinDrawText Parameter - clrBack

clrBack ([LONG](#)) - input
Background color.

The background is drawn with the current background mix. The default is BM_LEAVEALONE, that is, *clrBack* is ignored unless GpiSetBackMix is called.

The background rectangle is the rectangle that bounds the text; it is not the input parameter rectangle.

This parameter is ignored if DT_TEXTATTRS is specified.

WinDrawText Parameter - flCmd

flCmd ([ULONG](#)) - input
An array of flags that determines how the text is drawn.

Some of the DT_ flags are mutually exclusive. Only **one** from each of these groups is significant:

- DT_LEFT (default), DT_CENTER, DT_RIGHT
- DT_TOP (default), DT_VCENTER, DT_BOTTOM.

When mutually-exclusive flags are used together, the function gives indeterminate results.

If DT_HALFTONE, DT_ERASERECT, or DT_MNEMONIC is used, the presentation space must be in PU_PELS units.

DT_LEFT
Left-justify the text.

DT_CENTER
Center the text.

DT_RIGHT
Right-justify the text.

DT_VCENTER
Vertically center the text.

DT_TOP
Top-justify the text.

DT_BOTTOM
Bottom-justify the text.

DT_HALFTONE
Halftone the text display.

DT_MNEMONIC
If a mnemonic prefix character is encountered, the next character is drawn with mnemonic emphasis.

DT_QUERYEXTENT

The height *prc* is changed to a rectangle that bounds the string if it were drawn with WinDrawText.

DT_WORDBREAK

Only words that fit completely within the supplied rectangle are drawn. A *word* is defined as:

Any number of leading spaces followed by one or more visible characters and terminated by a space, carriage return, or line-feed character.

When calculating whether a particular word fits within the given rectangle, this function does not consider the trailing blanks. Only the length of the visible part of the word is tested against the right edge of the rectangle.

Also, note that this function always tries to draw at least one word, even if that word does not fit in the passed rectangle. This is so that progress is always made when drawing multiline text.

DT_EXTERNALLEADING

This flag causes the "external leading" value for the current font to be added to the bottom of the bounding rectangle before returning. It has an effect only when both DT_TOP and DT_QUERYEXTENT are also specified.

DT_TEXTATTRS

If this is specified, text is drawn using the character foreground and background colors of the presentation space, and *clrFore* and *clrBack* are ignored.

DT_ERASERECT

If this is specified, the rectangle defined by *prc* is erased before drawing the text. Otherwise, the background of the characters themselves can be erased if the character background mix (see "GpiSetAttrs" and "GpiSetBackMix" in the *Graphics Programming Interface Programming Reference*) is set to BM_OVERPAINT.

DT_UNDERSCORE

Underscore the characters. See FATTR_SEL_UNDERSCORE in the [FATTRS](#) datatype.

DT_STRIKEOUT

Overstrike the characters. See FATTR_SEL_STRIKEOUT in the [FATTRS](#) datatype.

WinDrawText Return Value - IChars

IChars ([LONG](#)) - returns

Count of characters drawn within the rectangle.

If DT_WORDBREAK is specified, this parameter returns the number of characters displayed. However, if the first word of the string does not fit in the rectangle, this parameter reflects the fact that the entire word is drawn.

If DT_WORDBREAK is **not** specified, the count returned is the full length of the string regardless of how much fits into the bounding rectangle.

0

Error occurred

Other

Count of characters drawn within the rectangle.

WinDrawText - Parameters

hps ([HPS](#)) - input

Presentation-space handle.

cchText ([LONG](#)) - input

Count of the number of characters in the string.

This parameter must be greater or equal to -1L.

-1L

The string is null-terminated and its length is to be calculated by this function.

Other

Count of the number of characters in the string.

lpchText (**PCH**) - input

Character string to be drawn.

A carriage-return or line-feed character always terminates a line. If the length of a line is less than *achText* (the string is divided into more than one line) each line is terminated by either of the mentioned characters.

prcl (**PRECTL**) - in/out

Text rectangle.

Rectangle within which the text is to be formatted, in world coordinates. Points on the boundary of this rectangle are deemed to be inside the rectangle.

On input, this rectangle is the desired rectangle in which the text is to be drawn.

On output, the height of the rectangle is modified to the actual size needed to draw the given text string. The return value is only of interest in the instance where DT_QUERYEXTENT is set in *flCmd*.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type **WRECT** can also be used, if supported by the language.

clrFore (**LONG**) - input

Foreground color.

Ignored if DT_TEXTATTRS is specified.

clrBack (**LONG**) - input

Background color.

The background is drawn with the current background mix. The default is BM_LEAVEALONE, that is, *clrBack* is ignored unless GpiSetBackMix is called.

The background rectangle is the rectangle that bounds the text; it is not the input parameter rectangle.

This parameter is ignored if DT_TEXTATTRS is specified.

flCmd (**ULONG**) - input

An array of flags that determines how the text is drawn.

Some of the DT_ flags are mutually exclusive. Only **one** from each of these groups is significant:

- DT_LEFT (default), DT_CENTER, DT_RIGHT
- DT_TOP (default), DT_VCENTER, DT_BOTTOM.

When mutually-exclusive flags are used together, the function gives indeterminate results.

If DT_HALFTONE, DT_ERASERECT, or DT_MNEMONIC is used, the presentation space must be in PU_PELS units.

DT_LEFT

Left-justify the text.

DT_CENTER

Center the text.

DT_RIGHT

Right-justify the text.

DT_VCENTER

Vertically center the text.

DT_TOP

Top-justify the text.

DT_BOTTOM

Bottom-justify the text.

DT_HALFTONE

Halftone the text display.

DT_MNEMONIC

If a mnemonic prefix character is encountered, the next character is drawn with mnemonic emphasis.

DT_QUERYEXTENT

The height *prc* is changed to a rectangle that bounds the string if it were drawn with WinDrawText.

DT_WORDBREAK

Only words that fit completely within the supplied rectangle are drawn. A *word* is defined as:

Any number of leading spaces followed by one or more visible characters and terminated by a space, carriage return, or line-feed character.

When calculating whether a particular word fits within the given rectangle, this function does not consider the trailing blanks. Only the length of the visible part of the word is tested against the right edge of the rectangle.

Also, note that this function always tries to draw at least one word, even if that word does not fit in the passed rectangle. This is so that progress is always made when drawing multiline text.

DT_EXTERNALLEADING

This flag causes the "external leading" value for the current font to be added to the bottom of the bounding rectangle before returning. It has an effect only when both DT_TOP and DT_QUERYEXTENT are also specified.

DT_TEXTATTRS

If this is specified, text is drawn using the character foreground and background colors of the presentation space, and *clrFore* and *clrBack* are ignored.

DT_ERASERECT

If this is specified, the rectangle defined by *prc* is erased before drawing the text. Otherwise, the background of the characters themselves can be erased if the character background mix (see "GpiSetAttrs" and "GpiSetBackMix" in the *Graphics Programming Interface Programming Reference*) is set to BM_OVERPAINT.

DT_UNDERSCORE

Underscore the characters. See FATTR_SEL_UNDERSCORE in the [FATTRS](#) datatype.

DT_STRIKEOUT

Overstrike the characters. See FATTR_SEL_STRIKEOUT in the [FATTRS](#) datatype.

IChars ([LONG](#)) - returns

Count of characters drawn within the rectangle.

If DT_WORDBREAK is specified, this parameter returns the number of characters displayed. However, if the first word of the string does not fit in the rectangle, this parameter reflects the fact that the entire word is drawn.

If DT_WORDBREAK is **not** specified, the count returned is the full length of the string regardless of how much fits into the bounding rectangle.

0

Error occurred

Other

Count of characters drawn within the rectangle.

WinDrawText - Remarks

Text is always drawn in the current font with the current foreground and background mix modes.

This function must only be used in draw mode (DM_DRAW), to a screen device context; *hps* can be either a micro presentation space or a normal presentation space (see "GpiCreatePS" in the *Graphics Programming Interface Programming Reference*).

WinDrawText - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_FLAG (0x1019)

An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

WinDrawText - Related Functions

Related Functions

- [WinDrawBitmap](#)
- [WinDrawBorder](#)
- [WinDrawPointer](#)
- [WinDrawText](#)
- [WinFillRect](#)
- [WinGetSysBitmap](#)
- [WinInvertRect](#)
- [WinQueryPresParam](#)
- [WinRemovePresParam](#)
- [WinScrollWindow](#)
- [WinSetPresParam](#)

WinDrawText - Example Code

This example shows how the WinDrawText function can be used to wrap text within a window by using the DT_WORDBREAK flag. The *cchDrawn* variable receives the number of characters actually drawn by the WinDrawText function. If this value is zero, no text is drawn and the for loop is exited. This can occur if the vertical height of the window is too short for the entire text. Otherwise, *cchDrawn* is added to the *hTotalDrawn* variable to provide an offset into the string for the next call to WinDrawText.

```
#define INCL_WINWINDOWMGR          /* Window Manager Functions */
#include <os2.h>

HWND    hwnd;                    /* parent window */
RECTL   rcl;                     /* update region */
HPS     hps;                     /* presentation-space handle */
char    *pszText;                /* string */
LONG     hText;                  /* length of string */
LONG     cyCharHeight;           /* set character height */
LONG     hTotalDrawn;            /* total characters drawn */
LONG     hDrawn;                 /* characters drawn by WinDrawText */
LONG     cchText;
LONG     cchTotalDrawn;
LONG     cchDrawn;

hps = WinGetPS(hwnd);            /* get a ps for the entire window */

WinQueryWindowRect(hwnd, &rcl);  /* get window dimensions */

WinFillRect(hps, &rcl, CLR_WHITE); /* clear entire window */

cchText = (LONG)strlen(pszText); /* get length of string */
cyCharHeight = 15L;              /* set character height */

/* until all chars drawn */
for (cchTotalDrawn = 0; hTotalDrawn != hText;
     rcl.yTop -= cyCharHeight)
{
    /* draw the text */

    hDrawn = WinDrawText(hps,      /* presentation-space handle */
                         hText - hTotalDrawn, /* length of text to draw */
                         pszText + hTotalDrawn, /* address of the text */
                         &rcl,      /* rectangle to draw in */

```

```

        0L,                                /* foreground color */
        0L,                                /* background color */
        DT_WORDBREAK | DT_TOP | DT_LEFT | DT_TEXTATTRS);
if (cchDrawn)
    hTotalDrawn += hDrawn;
else
    break;                                /* text could not be drawn */
}

WinReleasePS(hps);                        /* release the ps */

```

WinDrawText - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinEmptyClipbrd

WinEmptyClipbrd - Syntax

This function empties the clipboard, removing and freeing all handles to data that is in the clipboard.

```

#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB    hab; /* Anchor-block handle. */
BOOL    rc; /* Success indicator. */

rc = WinEmptyClipbrd(hab);

```

WinEmptyClipbrd Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinEmptyClipbrd Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinEmptyClipbrd - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinEmptyClipbrd - Remarks

The clipboard must be opened using [WinOpenClipbrd](#) before using this function.

This function will send a [WM_DESTROYCLIPBOARD](#) message to the clipboard owner.

WinEmptyClipbrd - Related Functions

Related Functions

- [WinCloseClipbrd](#)
 - [WinEmptyClipbrd](#)
 - [WinEnumClipbrdFmts](#)
 - [WinOpenClipbrd](#)
 - [WinQueryClipbrdData](#)
 - [WinQueryClipbrdFmtInfo](#)
 - [WinQueryClipbrdOwner](#)
 - [WinQueryClipbrdViewer](#)
 - [WinSetClipbrdData](#)
 - [WinSetClipbrdOwner](#)
 - [WinSetClipbrdViewer](#)
-

WinEmptyClipbrd - Example Code

This example empties the clipboard (opened by WinOpenClipbrd), removing and freeing all handles to data in the clipboard.

```
#define INCL_WINCLIPBOARD      /* Window Clipboard Functions */
#include <os2.h>

BOOL fSuccess;                /* success indicator */
HAB hab;                      /* anchor-block handle */

fSuccess = WinOpenClipbrd(hab);

if (fSuccess)
    fSuccess = WinEmptyClipbrd(hab);
```

WinEmptyClipbrd - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinEnableControl

WinEnableControl - Syntax

This macro sets the enable state of the item in the dialog template to the enable flag.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndDlg; /* Dialog window handle. */
USHORT  usId;    /* Identity of the item in the dialog template (button id). */
BOOL    fEnable; /* Enable flag. */
BOOL    rc;      /* Success indicator. */

rc = WinEnableControl(hwndDlg, usId, fEnable);
```

WinEnableControl Parameter - hwndDlg

hwndDlg (**HWND**) - input
Dialog window handle.

WinEnableControl Parameter - usId

usId (**USHORT**) - input
Identity of the item in the dialog template (button id).

WinEnableControl Parameter - fEnable

fEnable (**BOOL**) - input
Enable flag.

WinEnableControl Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinEnableControl - Parameters

hwndDlg (**HWND**) - input
Dialog window handle.

usId (**USHORT**) - input
Identity of the item in the dialog template (button id).

fEnable (**BOOL**) - input
Enable flag.

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinEnableControl - Remarks

This macro expands to:

```
#define WinEnableControl(hwndDlg, usId, fEnable)
WinEnableWindow(WinWindowFromId(hwndDlg, usId), fEnable)
```

This function requires the existence of a message queue.

WinEnableControl - Related Functions

Related Functions

- [WinEnableWindow](#)
 - [WinWindowFromID](#)
-

WinEnableControl - Example Code

This example uses WinEnableControl to enable a dialog control if it is currently disabled.

```
#define INCL_WINWINDOWMGR          /* Window Manager Functions */
#include <os2.h>

HWND      hwndDlg;                /* dialog window */
MPARAM    mp1;                   /* Parameter 1 */
USHORT    usId;                  /* dialog control id */

if (!WinIsControlEnabled(hwndDlg, usId))
    WinEnableControl(hwndDlg, usId, TRUE);
```

WinEnableControl - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinEnableMenuItem

WinEnableMenuItem - Syntax

This macro sets the state of the specified menu item to the enable flag.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND     hwndMenu; /* Menu window handle. */
USHORT   usId;     /* Item identifier. */
BOOL     fEnable;  /* Enable flag. */
BOOL     rc;       /* Success indicator. */

rc = WinEnableMenuItem(hwndMenu, usId, fEnable);
```

WinEnableMenuItem Parameter - hwndMenu

hwndMenu (**HWND**) - input
Menu window handle.

WinEnableMenuItem Parameter - usId

usId (**USHORT**) - input
Item identifier.

WinEnableMenuItem Parameter - fEnable

fEnable (**BOOL**) - input
Enable flag.

WinEnableMenuItem Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinEnableMenuItem - Parameters

hwndMenu ([HWND](#)) - input
Menu window handle.

usId ([USHORT](#)) - input
Item identifier.

fEnable ([BOOL](#)) - input
Enable flag.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinEnableMenuItem - Remarks

This macro expands to:

```
#define WinEnableMenuItem(hwndMenu, usId, fEnable)
((BOOL)WinSendMsg(hwndMenu,
    MM_SETITEMATTR,
    MPFROM2SHORT(usId, TRUE),
    MPFROM2SHORT(MIA_DISABLED, (BOOL)(fEnable) ? 0 : MIA_DISABLED)))
```

This function requires the existence of a message queue.

WinEnableMenuItem - Related Functions

Related Functions

- [WinSendMsg](#)
-

WinEnableMenuItem - Related Messages

Related Messages

- [MM_SETITEMATTR](#)

WinEnableMenuItem - Example Code

This example uses WinEnableMenuItem to make a menu item selection available when the menu is initialized (WM_INITMENU).

```
#define INCL_WINMESSAGEGR      /* Window Message Functions */
#define INCL_WINMENUS          /* Window Menu Functions */
#include <os2.h>

BOOL      fResult;           /* message-posted indicator */
MPARAM    mpParam1;          /* Parameter 1 (rectl structure) */
MPARAM    mpParam2;          /* Parameter 2 (frame boolean) */
USHORT    usItemId;          /* menu item id */
HWND      hwndMenu;          /* menu handle */

case WM_INITMENU:
    usItemId = SHORT1FROMMP(mpParam1);
    hwndMenu = HWNDFROMMP(mpParam2);

    /* enable menu item */
    fResult = WinEnableMenuItem(hwndMenu, usItemId, TRUE);
```

WinEnableMenuItem - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinEnablePhysInput

WinEnablePhysInput - Syntax

This function enables or disables queuing of physical input (keyboard or mouse).

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndDesktop; /* Desktop-window handle. */
```

```

BOOL    fEnable;        /* New state for the queuing of physical input. */
BOOL    rc;              /* Previous state for the queuing of physical input. */

rc = WinEnablePhysInput(hwndDesktop, fEnable);

```

WinEnablePhysInput Parameter - hwndDesktop

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

WinEnablePhysInput Parameter - fEnable

fEnable ([BOOL](#)) - input
New state for the queuing of physical input.

TRUE	Pointing device and keyboard input are queued
FALSE	Pointing device and keyboard input are disabled.

WinEnablePhysInput Return Value - rc

rc ([BOOL](#)) - returns
Previous state for the queuing of physical input.

TRUE	Pointing device and keyboard input were queued
FALSE	Pointing device and keyboard input were disabled.

WinEnablePhysInput - Parameters

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	

Specified desktop-window handle.

fEnable ([BOOL](#)) - input

New state for the queuing of physical input.

TRUE

Pointing device and keyboard input are queued

FALSE

Pointing device and keyboard input are disabled.

rc ([BOOL](#)) - returns

Previous state for the queuing of physical input.

TRUE

Pointing device and keyboard input were queued

FALSE

Pointing device and keyboard input were disabled.

WinEnablePhysInput - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)

An invalid window handle was specified.

WinEnablePhysInput - Related Functions

Related Functions

- [WinEnablePhysInput](#)
- [WinFocusChange](#)
- [WinGetKeyState](#)
- [WinGetPhysKeyState](#)
- [WinQueryFocus](#)
- [WinSetFocus](#)
- [WinSetKeyboardStateTable](#)

WinEnablePhysInput - Example Code

This example uses WinEnablePhysInput to enable queuing of physical input (pointing device and keyboard).

```
#define INCL_WININPUT          /* Window Input Functions */
#include <os2.h>

BOOL fOldInputState; /* previous queuing state */
BOOL fNewInputState=TRUE; /* new queuing state */

/* enable queuing of physical input */
fOldInputState = WinEnablePhysInput(HWND_DESKTOP, fNewInputState);
```

WinEnablePhysInput - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinEnableWindow

WinEnableWindow - Syntax

This function sets the window enabled state.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd;          /* Window handle. */
BOOL    fNewEnabled;    /* New enabled state. */
BOOL    rc;             /* Window enabled indicator. */

rc = WinEnableWindow(hwnd, fNewEnabled);
```

WinEnableWindow Parameter - hwnd

hwnd ([HWND](#)) - input
Window handle.

WinEnableWindow Parameter - fNewEnabled

fNewEnabled ([BOOL](#)) - input
New enabled state.

TRUE	Set window state to enabled
FALSE	Set window state to disabled.

WinEnableWindow Return Value - rc

rc ([BOOL](#)) - returns

Window enabled indicator.

TRUE

Window enabled state successfully updated

FALSE

Window enabled state not successfully updated.

WinEnableWindow - Parameters

hwnd ([HWND](#)) - input

Window handle.

fNewEnabled ([BOOL](#)) - input

New enabled state.

TRUE

Set window state to enabled

FALSE

Set window state to disabled.

rc ([BOOL](#)) - returns

Window enabled indicator.

TRUE

Window enabled state successfully updated

FALSE

Window enabled state not successfully updated.

WinEnableWindow - Remarks

If the enable state of *hwnd* is changing, a [WM_ENABLE](#) message is sent before this function returns.

If a window is disabled, its child windows are also disabled, although they are not sent the [WM_ENABLE](#) message. Typically, a window changes appearance when disabled. For example, a disabled push button is displayed with half-tone text.

If *hwnd* is disabled, and it, or one of its descendants, is the focus window, that window loses the focus, so that no window has the focus. However, a disabled window may subsequently be assigned the focus, in which case it should respond to keyboard input.

WinEnableWindow - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)

An invalid window handle was specified.

WinEnableWindow - Related Functions

Related Functions

- [WinEnableWindow](#)
 - [WinIsThreadActive](#)
 - [WinIsWindow](#)
 - [WinIsWindowEnabled](#)
 - [WinQueryDesktopWindow](#)
 - [WinQueryObjectWindow](#)
 - [WinQueryWindowDC](#)
 - [WinQueryWindowProcess](#)
 - [WinQueryWindowRect](#)
 - [WinWindowFromDC](#)
 - [WinWindowFromID](#)
 - [WinWindowFromPoint](#)
-

WinEnableWindow - Related Messages

Related Messages

- [WM_ENABLE](#)
-

WinEnableWindow - Example Code

This example uses `WinEnableWindow` to enable the system menu window for the given parent window, after verifying that the parent window handle is valid (`WinIsWindow`), belongs to the calling thread (`WinIsThreadActive`), and is not presently enabled (`WinIsWindowEnabled`).

```
#define INCL_WINWINDOWMGR          /* Window Manager Functions */
#define INCL_WINFRAMEMGR          /* Window Frame Functions */
#include <os2.h>

HAB    hab;           /* anchor-block handle */
HWND    hwndSystemMenu; /* system menu window */
HWND    hwnd;         /* parent window */
BOOL    fSuccess;     /* success indicator */

/* if handle specifies a valid window and the window belongs to the
   current thread, query the enabled status of the system menu */
if (WinIsWindow(hab, hwnd) && WinIsThreadActive(hab))
{
    /* obtain handle for system menu */
    hwndSystemMenu = WinWindowFromID(hwnd, FID_SYSMENU);

    /* if system menu is not enabled, enable it */
    if (!WinIsWindowEnabled(hwndSystemMenu))
        fSuccess = WinEnableWindow(hwndSystemMenu, TRUE);
}
```

WinEnableWindow - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Related Messages](#)

[Glossary](#)

WinEnableWindowUpdate

WinEnableWindowUpdate - Syntax

This function sets the window visibility state for subsequent drawing.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */  
#include <os2.h>
```

```
HWND    hwnd;        /* Window handle. */  
BOOL    fEnable;     /* New visibility state. */  
BOOL    rc;          /* Visibility-changed indicator. */
```

```
rc = WinEnableWindowUpdate(hwnd, fEnable);
```

WinEnableWindowUpdate Parameter - hwnd

hwnd (**HWND**) - input
Window handle.

WinEnableWindowUpdate Parameter - fEnable

fEnable (**BOOL**) - input
New visibility state.

TRUE

FALSE	Set window state visible
	Set window state invisible.

WinEnableWindowUpdate Return Value - rc

rc ([BOOL](#)) - returns
Visibility-changed indicator.

TRUE	Window visibility successfully changed
FALSE	Window visibility not successfully changed.

WinEnableWindowUpdate - Parameters

hwnd ([HWND](#)) - input
Window handle.

fEnable ([BOOL](#)) - input
New visibility state.

TRUE	Set window state visible
FALSE	Set window state invisible.

rc ([BOOL](#)) - returns
Visibility-changed indicator.

TRUE	Window visibility successfully changed
FALSE	Window visibility not successfully changed.

WinEnableWindowUpdate - Remarks

This function can be used to defer drawing when making a series of changes to a window. A window can be redrawn by using the [WinShowWindow](#) function.

WS_VISIBLE (style bit of a window) is set to *fEnable* without causing redrawing. That is, if the window was previously visible, it remains visible on the device when WS_VISIBLE is set to 0, and if the window was previously invisible, it is not shown when WS_VISIBLE is set to 1. If *fEnable* is set to TRUE, any subsequent drawing into the window is visible. If *fEnable* is set to FALSE, any subsequent drawing into the window is not visible.

If the value of the WS_VISIBLE style bit has been changed, the [WM_SHOW](#) message is sent to the window of *hwnd* before the call returns.

Any alteration to the appearance of a window disabled for window update is not presented. Therefore, the application must ensure that the window is redrawn. To show a window and ensure that it is redrawn after calling the WinEnableWindowUpdate function with *fEnable* set to FALSE, use the [WinShowWindow](#) function with *fNewVisibility* set to TRUE. In particular, if a window is destroyed while in this state its image is not removed from the display. After window updating is reenabled, the application should ensure that the window gets totally invalidated so that it repaints.

WinEnableWindowUpdate - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_INVALID_FLAG (0x1019)
An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

WinEnableWindowUpdate - Related Functions

Related Functions

- [WinBeginPaint](#)
 - [WinEnableWindowUpdate](#)
 - [WinEndPaint](#)
 - [WinExcludeUpdateRegion](#)
 - [WinGetClipPS](#)
 - [WinGetPS](#)
 - [WinGetScreenPS](#)
 - [WinInvalidateRect](#)
 - [WinInvalidateRegion](#)
 - [WinIsWindowShowing](#)
 - [WinIsWindowVisible](#)
 - [WinLockVisRegions](#)
 - [WinOpenWindowDC](#)
 - [WinQueryUpdateRect](#)
 - [WinQueryUpdateRegion](#)
 - [WinRealizePalette](#)
 - [WinReleasePS](#)
 - [WinShowWindow](#)
 - [WinUpdateWindow](#)
 - [WinValidateRect](#)
 - [WinValidateRegion](#)
-

WinEnableWindowUpdate - Related Messages

Related Messages

- [WM_PAINT](#)
-

WinEnableWindowUpdate - Example Code

This example uses WinEnableWindowUpdate to set a window's WS_VISIBLE style to visible and cause the window to be updated by a WM_PAINT message.

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#include <os2.h>
```

```

HWND    hwnd;           /* parent window          */
BOOL    fSuccess;       /* success indicator      */

case WM_CREATE:
    /* if window has WS_VISIBLE off, set state to visible */
    if (!WinIsWindowVisible(hwnd))
    {
        /* set state to visible and cause WM_PAINT message */
        fSuccess = WinEnableWindowUpdate(hwnd, EWUF_ENABLE);
    }

```

WinEnableWindowUpdate - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinEndEnumWindows

WinEndEnumWindows - Syntax

This function ends the enumeration process for a specified enumeration.

```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HENUM    henum; /* Enumeration handle. */
BOOL     rc;    /* Success indicator. */

rc = WinEndEnumWindows(henum);

```

WinEndEnumWindows Parameter - henum

henum (**HENUM**) - input
Enumeration handle.

Returned by previous call to the [WinBeginEnumWindows](#) call.

WinEndEnumWindows Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinEndEnumWindows - Parameters

henum ([HENUM](#)) - input
Enumeration handle.

Returned by previous call to the [WinBeginEnumWindows](#) call.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinEndEnumWindows - Remarks

This function destroys the window hierarchy remembered by the [WinBeginEnumWindows](#) function. After this function, the *henum* parameter is no longer valid.

WinEndEnumWindows - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HENUM (0x101C)
An invalid enumeration handle was specified.

WinEndEnumWindows - Related Functions

Related Functions

- [WinBeginEnumWindows](#)
- [WinEndEnumWindows](#)
- [WinEnumDlgItem](#)
- [WinGetNextWindow](#)
- [WinIsChild](#)
- [WinMultWindowFromIDs](#)
- [WinQueryWindow](#)
- [WinSetOwner](#)
- [WinSetParent](#)

WinEndEnumWindows - Example Code

This example ends the child window enumeration and releases the enumeration handle supplied by WinBeginEnumWindows after WinGetNextWindow has enumerated all immediate children of the Desktop.

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#include <os2.h>

HWND  hwndParent;           /* Handle of the window whose child windows
                             are to be enumerated */
HWND  hwndNext;             /* current enumeration handle */
HENUM  henum;               /* enumeration handle */
BOOL  fSuccess;             /* success indicator */
SHORT  sRetLen;             /* returned string length */
SHORT  sLength = 10;        /* string buffer length */
char  pchBuffer[10];        /* string buffer */

hwndParent = HWND_DESKTOP;

henum = WinBeginEnumWindows(hwndParent);

while ((hwndNext = WinGetNextWindow(henum)) != NULLHANDLE) {
    .
    .
    .
}
fSuccess = WinEndEnumWindows (henum);
```

WinEndEnumWindows - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Errors](#)
- [Remarks](#)
- [Example Code](#)
- [Related Functions](#)
- [Glossary](#)

WinEndPaint

WinEndPaint - Syntax

This function indicates that the redrawing of a window is complete, generally as part of the processing of a [WM_PAINT](#) message.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HPS    hps; /* Presentation-space handle. */
BOOL    rc; /* Success indicator. */

rc = WinEndPaint(hps);
```

WinEndPaint Parameter - hps

hps ([HPS](#)) - input
Presentation-space handle.

Handle of the presentation space that is used for drawing and that is returned by a previous call to the [WinBeginPaint](#) function.

WinEndPaint Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinEndPaint - Parameters

hps ([HPS](#)) - input
Presentation-space handle.

Handle of the presentation space that is used for drawing and that is returned by a previous call to the [WinBeginPaint](#) function.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinEndPaint - Remarks

The presentation space is restored to its state before the [WinBeginPaint](#) function:

- Cache presentation space is returned to the cache.
- Other presentation spaces have their original drawing state restored, including reassociating the original device context (if there was one).

If the pointer is hidden by the [WinBeginPaint](#) function, it is reshown by this function.

Any child windows having a synchronous painting style of the window associated with the presentation space are updated during the processing of this function, if they have non-NULL update regions.

WinEndPaint - Related Functions

Related Functions

- [WinBeginPaint](#)
 - [WinEnableWindowUpdate](#)
 - [WinEndPaint](#)
 - [WinExcludeUpdateRegion](#)
 - [WinGetClipPS](#)
 - [WinGetPS](#)
 - [WinGetScreenPS](#)
 - [WinInvalidateRect](#)
 - [WinInvalidateRegion](#)
 - [WinIsWindowShowing](#)
 - [WinIsWindowVisible](#)
 - [WinLockVisRegions](#)
 - [WinOpenWindowDC](#)
 - [WinQueryUpdateRect](#)
 - [WinQueryUpdateRegion](#)
 - [WinRealizePalette](#)
 - [WinReleasePS](#)
 - [WinShowWindow](#)
 - [WinUpdateWindow](#)
 - [WinValidateRect](#)
 - [WinValidateRegion](#)
-

WinEndPaint - Related Messages

Related Messages

- [WM_PAINT](#)
-

WinEndPaint - Example Code

This example uses [WinEndPaint](#) to end the update of a region and release the presentation space obtained by [WinBeginPaint](#).

```

#define INCL_WINWINDOWMGR          /* Window Manager Functions */
#include <os2.h>

HWND    hwnd;          /* parent window */
RECTL   rcl;           /* update region */
HPS     hps;           /* presentation-space handle */

case WM_PAINT:
    hps = WinBeginPaint(hwnd, /* handle of the window */
        NULLHANDLE,          /* get a cache presentation space */
        &rcl);                /* receives update rectangle */
    WinFillRect(hps, &rcl, CLR_WHITE);
    WinEndPaint(hps);

```

WinEndPaint - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinEnumClipbrdFmts

WinEnumClipbrdFmts - Syntax

This function enumerates the list of clipboard data formats available in the clipboard.

```

#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB    hab;          /* Anchor-block handle. */
ULONG   fmt;          /* Previous clipboard-data format index. */
ULONG   ulNext;       /* Next clipboard-data format index. */

ulNext = WinEnumClipbrdFmts(hab, fmt);

```

WinEnumClipbrdFmts Parameter - hab

hab ([HAB](#)) - input

Anchor-block handle.

WinEnumClipbrdFmts Parameter - fmt

fmt ([ULONG](#)) - input

Previous clipboard-data format index.

Specifies the index of the last clipboard data format enumerated using this function.

This should start at zero, in which instance the first available format is obtained. Subsequently, it should be set to the last format index value returned by this function.

WinEnumClipbrdFmts Return Value - ulNext

ulNext ([ULONG](#)) - returns

Next clipboard-data format index.

Possible values include:

- | | |
|-------|--|
| 0 | Enumeration is complete; that is, there are no more clipboard formats available. |
| Other | Index of the next available clipboard-data format in the clipboard. |
-

WinEnumClipbrdFmts - Parameters

hab ([HAB](#)) - input

Anchor-block handle.

fmt ([ULONG](#)) - input

Previous clipboard-data format index.

Specifies the index of the last clipboard data format enumerated using this function.

This should start at zero, in which instance the first available format is obtained. Subsequently, it should be set to the last format index value returned by this function.

ulNext ([ULONG](#)) - returns

Next clipboard-data format index.

Possible values include:

- | | |
|-------|--|
| 0 | Enumeration is complete; that is, there are no more clipboard formats available. |
| Other | Index of the next available clipboard-data format in the clipboard. |
-

WinEnumClipbrdFmts - Remarks

The clipboard should be open before this function is used.

WinEnumClipbrdFmts - Related Functions

Related Functions

- [WinCloseClipbrd](#)
 - [WinEmptyClipbrd](#)
 - [WinEnumClipbrdFmts](#)
 - [WinOpenClipbrd](#)
 - [WinQueryClipbrdData](#)
 - [WinQueryClipbrdFmtInfo](#)
 - [WinQueryClipbrdOwner](#)
 - [WinQueryClipbrdViewer](#)
 - [WinSetClipbrdData](#)
 - [WinSetClipbrdOwner](#)
 - [WinSetClipbrdViewer](#)
-

WinEnumClipbrdFmts - Example Code

This example enumerates and counts the available clipboard data formats for the clipboard opened by WinOpenClipbrd.

```
#define INCL_WINCLIPBOARD          /* Window Clipboard Functions */
#include <os2.h>

BOOL fSuccess;                    /* success indicator */
HAB hab;                          /* anchor-block handle */
ULONG ulNext;                    /* next format index */
ULONG ulPrev;                    /* previous format index */
ULONG ulNumFormats=0;            /* number of available formats */

fSuccess = WinOpenClipbrd(hab);

if (fSuccess)
{
    ulPrev = 0;
    /* enumerate formats and maintain count */
    while ((ulNext = WinEnumClipbrdFmts(hab, ulPrev)) != 0)
    {
        ulNumFormats++;
        ulPrev = ulNext;
    }
}
```

WinEnumClipbrdFmts - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Example Code](#)
- [Related Functions](#)

WinEnumDlgItem

WinEnumDlgItem - Syntax

This function returns the window handle of a dialog item within a dialog window.

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND     hwndDlg;    /* Dialog-window handle. */
HWND     hwnd;       /* Child-window handle. */
ULONG    code;       /* Item-type code. */
HWND     hwndItem;   /* Item-window handle. */

hwndItem = WinEnumDlgItem(hwndDlg, hwnd, code);
```

WinEnumDlgItem Parameter - hwndDlg

hwndDlg (**HWND**) - input
Dialog-window handle.

WinEnumDlgItem Parameter - hwnd

hwnd (**HWND**) - input
Child-window handle.

This may be an immediate child of the dialog window or a window lower in the window hierarchy, such as a child of a child window.

NULLHANDLE can be specified if *code* is EDI_FIRSTTABITEM or EDI_LASTTABITEM.

If an invalid handle is passed, the first child window of the dialog is returned.

WinEnumDlgItem Parameter - code

code (**ULONG**) - input
Item-type code.

Determines the type of dialog item to return. This parameter can have one of the following values:

EDI_FIRSTGROUPITEM
First item in the same group.

EDI_FIRSTTABITEM
First item in dialog with style **WS_TABSTOP**. *hwnd* is ignored.

EDI_LASTGROUPITEM
Last item in the same group.

EDI_LASTTABITEM
Last item in dialog with style **WS_TABSTOP**. *hwnd* is ignored.

EDI_NEXTGROUPITEM
Next item in the same group. Wraps around to beginning of group when the end of the group is reached.

EDI_NEXTTABITEM
Next item with style **WS_TABSTOP**. Wraps around to beginning of dialog item list when end is reached.

EDI_PREVGROUPITEM
Previous item in the same group. Wraps around to end of group when the start of the group is reached. For information on the **WS_GROUP** style,

EDI_PREVTABITEM
Previous item with style **WS_TABSTOP**. Wraps around to end of dialog item list when beginning is reached.

WinEnumDlgItem Return Value - hwndItem

hwndItem (**HWND**) - returns
Item-window handle.

As dictated by *code*.

The window is always an immediate child of *hwndDlg*, even if *hwnd* is not an immediate child window.

WinEnumDlgItem - Parameters

hwndDlg (**HWND**) - input
Dialog-window handle.

hwnd (**HWND**) - input
Child-window handle.

This may be an immediate child of the dialog window or a window lower in the window hierarchy, such as a child of a child window.

NULLHANDLE can be specified if *code* is **EDI_FIRSTTABITEM** or **EDI_LASTTABITEM**.

If an invalid handle is passed, the first child window of the dialog is returned.

code (**ULONG**) - input
Item-type code.

Determines the type of dialog item to return. This parameter can have one of the following values:

EDI_FIRSTGROUPITEM

First item in the same group.

EDI_FIRSTTABITEM

First item in dialog with style WS_TABSTOP. *hwnd* is ignored.

EDI_LASTGROUPITEM

Last item in the same group.

EDI_LASTTABITEM

Last item in dialog with style WS_TABSTOP. *hwnd* is ignored.

EDI_NEXTGROUPITEM

Next item in the same group. Wraps around to beginning of group when the end of the group is reached.

EDI_NEXTTABITEM

Next item with style WS_TABSTOP. Wraps around to beginning of dialog item list when end is reached.

EDI_PREVGROUPITEM

Previous item in the same group. Wraps around to end of group when the start of the group is reached. For information on the WS_GROUP style,

EDI_PREVTABITEM

Previous item with style WS_TABSTOP. Wraps around to end of dialog item list when beginning is reached.

hwndItem ([HWND](#)) - returns
Item-window handle.

As dictated by *code*.

The window is always an immediate child of *hwndDlg*, even if *hwnd* is not an immediate child window.

WinEnumDlgItem - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinEnumDlgItem - Related Functions

Related Functions

- [WinBeginEnumWindows](#)
- [WinEndEnumWindows](#)
- [WinEnumDlgItem](#)
- [WinGetNextWindow](#)
- [WinIsChild](#)
- [WinMultWindowFromIDs](#)
- [WinQueryWindow](#)
- [WinSetOwner](#)
- [WinSetParent](#)

WinEnumDlgItem - Example Code

This example uses WinEnumDlgItem to query the first dialog item for each immediate child of the specified dialog window. The immediate children are enumerated using a WinBeginEnumWindows - WinGetNextWindow - WinEndEnumWindows loop.

```
#define INCL_WINDIALOGS      /* Window Dialog Mgr functions */
#define INCL_WINWINDOWMGR   /* Window Manager functions */
#include <os2.h>

HWND  hwndDlg;           /* Handle of the parent dialog window */
HWND  hwndChild;         /* Current dialog child */
HWND  hwndItem;          /* First dialog item */
HENUM henum;             /* Enumeration handle */
BOOL  fSuccess;          /* Success indicator */

henum = WinBeginEnumWindows(hwndDlg);
while ((hwndChild = WinGetNextWindow(henum)) != NULL)
    hwndItem = WinEnumDlgItem(hwndDlg, hwndChild, EDI_FIRSTTABITEM);
fSuccess = WinEndEnumWindows(henum);
```

WinEnumDlgItem - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinEnumObjectClasses

WinEnumObjectClasses - Syntax

The WinEnumObjectClasses function will return a list of all workplace object classes that have been registered.

```
#define INCL_WINWORKPLACE
#include <os2.h>

POBJCLASS  pObjClass; /* Pointer to object class. */
PULONG     pSize;     /* Length of the pObjClass buffer in bytes. */
BOOL       rc;        /* Success indicator. */

rc = WinEnumObjectClasses(pObjClass, pSize);
```

WinEnumObjectClasses Parameter - pObjClass

pObjClass ([POBJCLASS](#)) - input
Pointer to object class.

A pointer to a buffer to be filled with information about the registered workplace object classes.

WinEnumObjectClasses Parameter - pSize

pSize ([PULONG](#)) - in/out
Length of the *pObjClass* buffer in bytes.

If *pObjClass* is NULL, the actual size of *pObjClass* is returned in *pSize*

WinEnumObjectClasses Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinEnumObjectClasses - Parameters

pObjClass ([POBJCLASS](#)) - input
Pointer to object class.

A pointer to a buffer to be filled with information about the registered workplace object classes.

pSize ([PULONG](#)) - in/out
Length of the *pObjClass* buffer in bytes.

If *pObjClass* is NULL, the actual size of *pObjClass* is returned in *pSize*

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinEnumObjectClasses - Remarks

WinEnumObjectClasses will return a buffer containing all workplace object classes that are currently registered with the system. Workplace object classes are registered with the system through the function call [WinRegisterObjectClass](#).

WinEnumObjectClasses - Related Functions

Related Functions

- [WinRegisterObjectClass](#)
- [WinReplaceObjectClass](#)

WinEnumObjectClasses - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

WinEqualRect

WinEqualRect - Syntax

This function compares two rectangles for equality.

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;      /* Anchor-block handle. */
PRECTL   prcl1;    /* First rectangle. */
PRECTL   prcl2;    /* Second rectangle. */
BOOL      rc;       /* Equality indicator. */

rc = WinEqualRect(hab, prcl1, prcl2);
```

WinEqualRect Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinEqualRect Parameter - prcl1

prcl1 ([PRECTL](#)) - input
First rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

WinEqualRect Parameter - prcl2

prcl2 ([PRECTL](#)) - input
Second rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

WinEqualRect Return Value - rc

rc ([BOOL](#)) - returns
Equality indicator.

TRUE	Rectangles are identical
FALSE	Rectangles are not identical, or an error occurred.

WinEqualRect - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

prcl1 ([PRECTL](#)) - input
First rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

prcl2 ([PRECTL](#)) - input

Second rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

rc ([BOOL](#)) - returns
Equality indicator.

TRUE

Rectangles are identical

FALSE

Rectangles are not identical, or an error occurred.

WinEqualRect - Remarks

If both rectangles are empty (for example, they are considered equal even if the actual coordinate values are different).

WinEqualRect - Related Functions

Related Functions

- [WinCopyRect](#)
- [WinEqualRect](#)
- [WinFillRect](#)
- [WinInflateRect](#)
- [WinIntersectRect](#)
- [WinIsRectEmpty](#)
- [WinOffsetRect](#)
- [WinPtInRect](#)
- [WinSetRect](#)
- [WinSetRectEmpty](#)
- [WinSubtractRect](#)
- [WinUnionRect](#)

WinEqualRect - Example Code

This example compares two rectangles for equality.

```
#define INCL_WINRECTANGLES      /* Window Rectangle Functions */
#include <os2.h>

BOOL fEqual;                    /* equal indicator */
HAB hab;                       /* anchor-block handle */
RECTL prclRect1 = {0,0,100,100}; /* first rectangle */
RECTL prclRect2 = {0,0,200,200}; /* second rectangle */

fEqual = WinEqualRect(hab, &prclRect1, &prclRect2);
```

WinEqualRect - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinExcludeUpdateRegion

WinExcludeUpdateRegion - Syntax

This function subtracts the update region (invalid region) of a window from the clipping region of a presentation space.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HPS      hps;           /* Presentation-space handle whose clipping region is to be updated. */
HWND     hwnd;          /* Window handle. */
LONG     lComplexity;    /* Complexity value. */

lComplexity = WinExcludeUpdateRegion(hps,
                                     hwnd);
```

WinExcludeUpdateRegion Parameter - hps

hps ([HPS](#)) - input

Presentation-space handle whose clipping region is to be updated.

WinExcludeUpdateRegion Parameter - hwnd

hwnd ([HWND](#)) - input

Window handle.

Handle of window whose update region is subtracted from the clipping region of the presentation space.

WinExcludeUpdateRegion Return Value - lComplexity

lComplexity ([LONG](#)) - returns
Complexity value.

This indicates the resulting form of the clipping area. The values and meanings of this parameter are defined in `GpiCombineRegion`.

Complexity of resulting region/error indicator:

<code>RGN_NULL</code>	Null Region
<code>RGN_RECT</code>	Rectangle region
<code>RGN_COMPLEX</code>	Complex region
<code>RGN_ERROR</code>	Error.

WinExcludeUpdateRegion - Parameters

hps ([HPS](#)) - input
Presentation-space handle whose clipping region is to be updated.

hwnd ([HWND](#)) - input
Window handle.

Handle of window whose update region is subtracted from the clipping region of the presentation space.

lComplexity ([LONG](#)) - returns
Complexity value.

This indicates the resulting form of the clipping area. The values and meanings of this parameter are defined in `GpiCombineRegion`.

Complexity of resulting region/error indicator:

<code>RGN_NULL</code>	Null Region
<code>RGN_RECT</code>	Rectangle region
<code>RGN_COMPLEX</code>	Complex region
<code>RGN_ERROR</code>	Error.

WinExcludeUpdateRegion - Remarks

This function is typically used to prevent drawing into parts of a window that are known to be invalid, as during an incremental update optimization process.

It is the application's responsibility to reset the clipping region when necessary.

WinExcludeUpdateRegion - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinExcludeUpdateRegion - Related Functions

Related Functions

- [WinBeginPaint](#)
 - [WinEnableWindowUpdate](#)
 - [WinEndPaint](#)
 - [WinExcludeUpdateRegion](#)
 - [WinGetClipPS](#)
 - [WinGetPS](#)
 - [WinGetScreenPS](#)
 - [WinInvalidateRect](#)
 - [WinInvalidateRegion](#)
 - [WinIsWindowShowing](#)
 - [WinIsWindowVisible](#)
 - [WinLockVisRegions](#)
 - [WinOpenWindowDC](#)
 - [WinQueryUpdateRect](#)
 - [WinQueryUpdateRegion](#)
 - [WinRealizePalette](#)
 - [WinReleasePS](#)
 - [WinShowWindow](#)
 - [WinUpdateWindow](#)
 - [WinValidateRect](#)
 - [WinValidateRegion](#)
-

WinExcludeUpdateRegion - Example Code

This example uses `WinExcludeUpdateRegion` to prevent drawing into the window's known invalid regions (to optimize updates) by excluding the window's update region from the clipping region of the presentation space. The clipping region will need to be reset later by the application, which can be accomplished using `GpiIntersectClipRectangle` with the rectangle comprising the window as input.

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#define INCL_GPIREGIONS      /* Region functions */
#include <os2.h>

LONG lComplexity;      /* clipping complexity/error return */
HWND hwnd;            /* parent window */
RECT rcl;              /* update region */
HPS hps;              /* presentation-space handle */

case WM_PAINT:
    lComplexity = WinExcludeUpdateRegion(hps, hwnd);

    hps = WinBeginPaint(hwnd, /* handle of the window */
        NULLHANDLE,          /* get a cache presentation space */
        &rcl);                /* receives update rectangle */
    WinFillRect(hps, &rcl, CLR_WHITE);
    WinEndPaint(hps);
```

WinExcludeUpdateRegion - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinFileDlg

WinFileDlg - Syntax

This function creates and displays the file dialog and returns the user's selection or selections.

```
#define INCL_winstdfile
#include <os2.h>

HWND      hwndP;      /* Parent-window handle. */
HWND      hwndO;      /* Requested owner-window handle. */
PFILEDLG  pfild;      /* Pointer to a FILEDLG structure. */
HWND      hwndDlg;    /* File dialog window handle. */

hwndDlg = WinFileDlg(hwndP, hwndO, pfild);
```

WinFileDlg Parameter - hwndP

hwndP ([HWND](#)) - input

Parent-window handle.

Parent-window handle of the created dialog window.

HWND_DESKTOP

The desktop window.

Other

Specified window.

WinFileDlg Parameter - hwndO

hwndO ([HWND](#)) - input

Requested owner-window handle.

Requested owner-window handle of the created dialog window.

The actual owner window is calculated using the algorithm specified in the description of the [WinLoadDlg](#) function.

WinFileDlg Parameter - pfiled

pfiled ([PFILEDLG](#)) - input
Pointer to a [FILEDLG](#) structure.

WinFileDlg Return Value - hwndDlg

hwndDlg ([HWND](#)) - returns
File dialog window handle.

If the flag is set by the application, the return value is the window handle of the file dialog, or NULLHANDLE if the dialog cannot be created. If the flag is not set, the return value is TRUE if dialog creation is successful, or NULLHANDLE if it is unsuccessful.

WinFileDlg - Parameters

hwndP ([HWND](#)) - input
Parent-window handle.

Parent-window handle of the created dialog window.

HWND_DESKTOP
The desktop window.

Other
Specified window.

hwndO ([HWND](#)) - input
Requested owner-window handle.

Requested owner-window handle of the created dialog window.

The actual owner window is calculated using the algorithm specified in the description of the [WinLoadDlg](#) function.

pfiled ([PFILEDLG](#)) - input
Pointer to a [FILEDLG](#) structure.

hwndDlg ([HWND](#)) - returns
File dialog window handle.

If the flag is set by the application, the return value is the window handle of the file dialog, or NULLHANDLE if the dialog cannot be created. If the flag is not set, the return value is TRUE if dialog creation is successful, or NULLHANDLE if it is unsuccessful.

WinFileDlg - Remarks

The *pfild* parameter is required and the [FILEDLG](#) structure must be properly initialized.

On return, the [FILEDLG](#) structure is updated with any user alterations, and the field is set to the value returned by the file dialog's [WinDismissDlg](#) function. By default, this is the ID of the push button pressed to dismiss the dialog, DID_OK or DID_CANCEL, unless the application supplied additional push buttons in its template.

For convenience, the pointer to the [FILEDLG](#) structure is placed in the QWL_USER field of the dialog's frame window. If in a custom file dialog procedure the pointer to the [FILEDLG](#) structure is desired, it should be queried from the frame window with the [WinQueryWindowULong](#) function.

To subclass the default file dialog with a new template, the application must give the module and ID of the new file dialog template and the address of a dialog procedure for message handling. Window IDs in the range 0x0000 through 0x0FFF are reserved for the standard file dialog controls. IDs from outside this range must be chosen for any controls or windows added to a custom file dialog.

When a modeless dialog is dismissed, the owner of the file dialog will receive a [WM_COMMAND](#) message with the the ID of the file dialog.

WinFileDlg - Example Code

This example uses WinFileDlg to create and display a single file selection dialog using the system default open file dialog template and procedure.

```
#define INCL_WINSTDFILE          /* Window Standard File      */
                                /* functions                    */
#include <os2.h>

FILEDLG fild;                  /* File dialog info structure */
char pszTitle[10] = "Open File"; /* Title of dialog           */
char pszFullFile[CCHMAXPATH] = "*.C"; /* File filter string        */
HWND hwndMain;                 /* Window that owns the file */
                                /* dialog                     */
HWND hwndDlg;                  /* File dialog window        */

/* Initially set all fields to 0 */
memset(&pfidFiledlg, 0, sizeof(FILEDLG));

/* Initialize those fields in the FILEDLG structure */
/* that are used by the application */
fild.cbSize = sizeof(FILEDLG); /* Size of structure         */
fild.fl = FDS_HELPBUTTON |    /* FDS_* flags               */
         FDS_CENTER |
         FDS_OPEN_DIALOG;
fild.pszTitle = pszTitle;      /* Dialog title string       */

/* Initial path, file name, or file filter */
strcpy(fild.szFullFile, pszFullFile);

/* Display the dialog and get the file */
hwndDlg = WinFileDlg(HWND_DESKTOP, hwndMain, &fild);
if (hwndDlg && (pfild.lReturn == DID_OK))
{
    /* Upon successful return of a file, open it for reading */
    /* and further processing */
}
```

WinFileDlg - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

WinFillRect

WinFillRect - Syntax

This function draws a filled rectangular area.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HPS      hps;      /* Presentation-space handle. */
PRECTL   prcl;     /* Rectangle to be filled, in window coordinates. */
LONG     lColor;    /* Color with which to fill the rectangle. */
BOOL     rc;        /* Success indicator. */

rc = WinFillRect(hps, prcl, lColor);
```

WinFillRect Parameter - hps

hps ([HPS](#)) - input
Presentation-space handle.

This can be either a micro-presentation space or a normal presentation space.

WinFillRect Parameter - prcl

prcl ([PRECTL](#)) - input
Rectangle to be filled, in window coordinates.

Points on the left and bottom boundaries of the rectangle are included in the fill, but points on the right and top boundaries are not, except where they are also on the left and bottom boundaries; that is, the top-left and bottom-right corners.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

WinFillRect Parameter - lColor

IColor ([LONG](#)) - input
Color with which to fill the rectangle.

This is either a color index, or an RGB color value, depending upon whether and how a logical color table has been loaded. (See "GdiCreateLogColorTable" in the *Graphics Programming Interface Programming Reference*.)

WinFillRect Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinFillRect - Parameters

hps ([HPS](#)) - input
Presentation-space handle.

This can be either a micro-presentation space or a normal presentation space.

prcl ([PRECTL](#)) - input
Rectangle to be filled, in window coordinates.

Points on the left and bottom boundaries of the rectangle are included in the fill, but points on the right and top boundaries are not, except where they are also on the left and bottom boundaries; that is, the top-left and bottom-right corners.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

IColor ([LONG](#)) - input
Color with which to fill the rectangle.

This is either a color index, or an RGB color value, depending upon whether and how a logical color table has been loaded. (See "GdiCreateLogColorTable" in the *Graphics Programming Interface Programming Reference*.)

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinFillRect - Remarks

This function does not change any presentation space state.

This function must only be used in draw mode (DM_DRAW) to a screen device context.

If an empty rectangle is specified, this function draws nothing and completes successfully (that is, TRUE is returned).

WinFillRect - Related Functions

Related Functions

- [WinCopyRect](#)
- [WinDrawBitmap](#)
- [WinDrawBorder](#)
- [WinDrawPointer](#)
- [WinDrawText](#)
- [WinEqualRect](#)
- [WinGetSysBitmap](#)
- [WinInflateRect](#)
- [WinIntersectRect](#)
- [WinIsRectEmpty](#)
- [WinInvertRect](#)
- [WinOffsetRect](#)
- [WinPtInRect](#)
- [WinQueryPresParam](#)
- [WinRemovePresParam](#)
- [WinScrollWindow](#)
- [WinSetPresParam](#)
- [WinSetRect](#)
- [WinSetRectEmpty](#)
- [WinSubtractRect](#)
- [WinUnionRect](#)

WinFillRect - Example Code

This example fills an update rectangle with a white background in response to the WM_PAINT message, after obtaining a presentation space handle via WinBeginPaint.

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#include <os2.h>

BOOL fSuccess;                /* success indicator */
HAB hab;                      /* anchor-block handle */
PRECTL prclRect1 = {0,0,100,100}; /* fill rectangle */
LONG lColor=CLR_WHITE;        /* fill color */
HWND hwnd;                   /* client window handle */
HPS hps;                     /* presentation-space handle */

case WM_PAINT:
    hps = WinBeginPaint(hwnd, NULLHANDLE, &prclRect1);
    fSuccess = WinFillRect(hps, &prclRect1, lColor);
    WinEndPaint(hps);
```

WinFillRect - Topics

Select an item:

[Syntax](#)

[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinFindAtom

WinFindAtom - Syntax

This function finds an atom in the atom table.

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HATOMTBL    hatomtblAtomTbl; /* Atom-table handle. */
PSZ         pszAtomName;     /* Atom name. */
ATOM        atom;             /* Atom value. */

atom = WinFindAtom(hatomtblAtomTbl, pszAtomName);
```

WinFindAtom Parameter - hatomtblAtomTbl

hatomtblAtomTbl ([HATOMTBL](#)) - input
Atom-table handle.

This is the handle returned from a previous [WinCreateAtomTable](#) or [WinQuerySystemAtomTable](#) function.

WinFindAtom Parameter - pszAtomName

pszAtomName ([PSZ](#)) - input
Atom name.

This is a null terminated character string to be found in the table.

If the string begins with a "#" character, the five ASCII digits that follow are converted into an integer atom.

If the string begins with a "!" character, the next two bytes are interpreted as an atom.

If the high order word of the string is -1, the low order word is an atom.

WinFindAtom Return Value - atom

atom ([ATOM](#)) - returns
Atom value.

Atom	The atom associated with the passed string
0	Invalid atom table handle or invalid atom name specified.

WinFindAtom - Parameters

hatomtblAtomTbl ([HATOMTBL](#)) - input
Atom-table handle.

This is the handle returned from a previous [WinCreateAtomTable](#) or [WinQuerySystemAtomTable](#) function.

pszAtomName ([PSZ](#)) - input
Atom name.

This is a null terminated character string to be found in the table.

If the string begins with a "#" character, the five ASCII digits that follow are converted into an integer atom.

If the string begins with a "!" character, the next two bytes are interpreted as an atom.

If the high order word of the string is -1, the low order word is an atom.

atom ([ATOM](#)) - returns
Atom value.

Atom	The atom associated with the passed string
0	Invalid atom table handle or invalid atom name specified.

WinFindAtom - Remarks

This function is identical to the [WinAddAtom](#) function, except that:

- If the atom name is not found in the table, it is not added to the table and 0 is returned.
- If the atom name is found in the table, the use count is not incremented.

Because integer atoms do not have a use count and do not actually occupy memory in the atom table, this function is identical to [WinAddAtom](#) with respect to integer atoms.

WinFindAtom - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HATOMTBL (0x1013)
An invalid atom-table handle was specified.

PMERR_INVALID_INTEGER_ATOM (0x1016)
The specified atom is not a valid integer atom.

PMERR_INVALID_ATOM_NAME (0x1015)
An invalid atom name string was passed.

PMERR_ATOM_NAME_NOT_FOUND (0x1017)
The specified atom name is not in the atom table.

WinFindAtom - Related Functions

Related Functions

- [WinAddAtom](#)
- [WinCreateAtomTable](#)
- [WinDeleteAtom](#)
- [WinDestroyAtomTable](#)
- [WinFindAtom](#)
- [WinQueryAtomLength](#)
- [WinQueryAtomName](#)
- [WinQueryAtomUsage](#)
- [WinQuerySystemAtomTable](#)

WinFindAtom - Example Code

This example queries an Atom Table for the atom name of a newly created atom "newatom" and then verifies that the atom value returned by the query matches the atom value returned by WinAddAtom.

```
#define INCL_WINATOM          /* Window Atom Functions */
#include <os2.h>

ATOM atom;                  /* new atom value */
ATOM atomFound;             /* atom value from WinFindAtom */
HATOMTBL hatomtblAtomTbl;   /* atom-table handle */
char pszAtomName[10];       /* atom name */
ULONG ulInitial = 0;        /* initial atom table size (use default) */
ULONG ulBuckets = 0;        /* size of hash table (use default) */
BOOL atomMatch = FALSE;     /* indicates atom values match */

/* create atom table of default size */
hatomtblAtomTbl = WinCreateAtomTable(ulInitial, ulBuckets);

/* define name for new atom and add to table */
strcpy(pszAtomName, "newatom");
atom = WinAddAtom(hatomtblAtomTbl, pszAtomName);

atomFound = WinFindAtom(hatomtblAtomTbl, pszAtomName);

/* verify that the atom values match */
if (atom == atomFound)
    atomMatch = TRUE;
```

WinFindAtom - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinFlashWindow

WinFlashWindow - Syntax

This function starts or stops a window flashing.

```
#define INCL_WINFRAMEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndFrame; /* Handle of window to be flashed. */
BOOL    fFlash;    /* Start-flashing indicator. */
BOOL    rc;         /* Success indicator. */

rc = WinFlashWindow(hwndFrame, fFlash);
```

WinFlashWindow Parameter - hwndFrame

hwndFrame ([HWND](#)) - input
Handle of window to be flashed.

WinFlashWindow Parameter - fFlash

fFlash ([BOOL](#)) - input
Start-flashing indicator.

TRUE	Start window flashing
FALSE	Stop window flashing.

WinFlashWindow Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinFlashWindow - Parameters

hwndFrame ([HWND](#)) - input
Handle of window to be flashed.

fFlash ([BOOL](#)) - input
Start-flashing indicator.

TRUE	Start window flashing
FALSE	Stop window flashing.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinFlashWindow - Remarks

Flashing a window brings the user's attention to a window that is not the active window, where some important message or dialog must be seen by the user.

Flashing is typically done by inverting the title bar continuously. The alarm is sounded for the first five flashes.

Note: It should be used only for important messages, for example, where some component of the system is failing and requires immediate attention to avoid damage.

WinFlashWindow - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)

An invalid window handle was specified.

WinFlashWindow - Related Functions

Related Functions

- [WinAlarm](#)
 - [WinFlashWindow](#)
 - [WinMessageBox](#)
-

WinFlashWindow - Example Code

This example uses WinFlashWindow to flash an inactive window to draw the user's attention to an important message in the window.

```
#define INCL_WINFRAMEMGR      /* Window Frame Functions */
#define INCL_WINDIALOGS      /* Window Dialog Mgr Functions */
#include <os2.h>

BOOL fSuccess;               /* Success indicator */
HWND hwnd;                  /* window handle */

/* flash window to get user's attention */
fSuccess = WinFlashWindow(hwnd, TRUE);

/* vital message is displayed */
WinMessageBox(HWND_DESKTOP,
    hwnd,                  /* client-window handle */
    "Important message: must be seen by user", /* message */
    "Vital message",       /* title of the message */
    0,                     /* message box id */
    MB_NOICON | MB_OK);    /* icon and button flags */
```

WinFlashWindow - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinFocusChange

WinFocusChange - Syntax

This function changes the focus window.

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND     hwndDesktop;    /* Desktop-window handle. */
HWND     hwndNewFocus;   /* Window handle to receive the focus. */
ULONG    flFocusChange;  /* Focus changing indicators. */
BOOL     rc;             /* Success indicator. */

rc = WinFocusChange(hwndDesktop, hwndNewFocus,
                    flFocusChange);
```

WinFocusChange Parameter - hwndDesktop

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

WinFocusChange Parameter - hwndNewFocus

hwndNewFocus ([HWND](#)) - input
Window handle to receive the focus.

WinFocusChange Parameter - flFocusChange

flFocusChange ([ULONG](#)) - input
Focus changing indicators.

These indicators are passed on in the [WM_FOCUSCHANGE](#) message:

FC_NOSETFOCUS	Do not send the WM_SETFOCUS message to the window receiving the focus.
---------------	--

FC_NOLOSEFOCUS	Do not send the WM_SETFOCUS message to the window losing the focus.
----------------	---

FC_NOSETACTIVE	
----------------	--

Do not send the [WM_ACTIVATE](#) message to the window being activated.

FC_NOLOSEACTIVE

Do not send the [WM_ACTIVATE](#) message to the window being deactivated.

FC_NOSETSELECTION

Do not send the [WM_SETSELECTION](#) message to the window being selected.

FC_NOLOSESELECTION

Do not send the [WM_SETSELECTION](#) message to the window being deselected.

FC_NOBRINGTOTOP

Do not bring any window to the top.

FC_NOBRINGTOTOPFIRSTWINDOW

Do not bring the first frame window to the top.

WinFocusChange Return Value - rc

rc ([BOOL](#)) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinFocusChange - Parameters

hwndDeskTop ([HWND](#)) - input

Desktop-window handle.

HWND_DESKTOP

The desktop-window handle

Other

Specified desktop-window handle.

hwndNewFocus ([HWND](#)) - input

Window handle to receive the focus.

flFocusChange ([ULONG](#)) - input

Focus changing indicators.

These indicators are passed on in the [WM_FOCUSCHANGE](#) message:

FC_NOSETFOCUS

Do not send the [WM_SETFOCUS](#) message to the window receiving the focus.

FC_NOLOSEFOCUS

Do not send the [WM_SETFOCUS](#) message to the window losing the focus.

FC_NOSETACTIVE

Do not send the [WM_ACTIVATE](#) message to the window being activated.

FC_NOLOSEACTIVE

Do not send the [WM_ACTIVATE](#) message to the window being deactivated.

FC_NOSETSELECTION

Do not send the [WM_SETSELECTION](#) message to the window being selected.

FC_NOLOSESELECTION

Do not send the [WM_SETSELECTION](#) message to the window being deselected.

FC_NOBRINGTOTOP

Do not bring any window to the top.

FC_NOBRINGTOTOPFIRSTWINDOW

Do not bring the first frame window to the top.

rc ([BOOL](#)) - returns
Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinFocusChange - Remarks

This function sends a [WM_FOCUSCHANGE](#) message to the window that is losing the focus and a [WM_FOCUSCHANGE](#) message to the window that is receiving the focus.

This function fails if another process or thread is currently using this function.

Other messages may be sent as a consequence of the frame control processing of the [WM_FOCUSCHANGE](#) (in [Frame Controls](#)) message, depending on the value of the *//FocusChange* parameter. These messages, if sent, are sent in this order:

1. [WM_SETFOCUS](#) to the window losing the focus.
2. [WM_SETSELECTION](#) to the windows losing their selection.
3. [WM_ACTIVATE](#) to the windows being deactivated.
4. [WM_ACTIVATE](#) to the windows being activated.
5. [WM_SETSELECTION](#) to the windows being selected.
6. [WM_SETFOCUS](#) to the window receiving the focus.

Note: If the [WinQueryFocus](#) function is used during processing of this function:

- The window handle of the window losing the focus is returned while the [WM_FOCUSCHANGE](#) message with the is being processed.
- The window handle of the window receiving the focus is returned while the [WM_FOCUSCHANGE](#) (in [Frame Controls](#)) message with the is being processed.

If the [WinQueryActiveWindow](#) function is used during processing of this function:

- The window handle of the window being deactivated is returned while the [WM_ACTIVATE](#) message with the is being processed.
- The window handle of the window being activated is returned while the [WM_ACTIVATE](#) message with the is being processed.

Also, there is a short period during the time after the old active window has acted on the deactivation message and before the new active window has acted on the activation message when the [WinQueryActiveWindow](#) function returns `NULLHANDLE`.

This function should not be made unless it is directly or indirectly the result of operator input.

Even if `FC_NOSETSELECTION` is not specified, the [WM_SETSELECTION](#) is not sent to a frame window that is already selected. This can occur if the focus is being transferred from a parent to a child window and `FC_NOLOSESELECTION` was specified.

WinFocusChange - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinFocusChange - Related Functions

Related Functions

- [WinEnablePhysInput](#)
 - [WinFocusChange](#)
 - [WinGetKeyState](#)
 - [WinGetPhysKeyState](#)
 - [WinQueryFocus](#)
 - [WinSetFocus](#)
 - [WinSetKeyboardStateTable](#)
-

WinFocusChange - Related Messages

Related Messages

- [WM_ACTIVATE](#)
 - [WM_FOCUSCHANGE](#)
 - [WM_SETFOCUS](#)
 - [WM_SETSELECTION](#)
-

WinFocusChange - Example Code

This example uses WinFocusChange to change the focus to the selected window, using the handle returned by WinQueryFocus.

```
#define INCL_WININPUT          /* Window Input Functions          */
#include <os2.h>

HWND  hwndNewFocus;           /* Handle of new focus window          */
BOOL  fSuccess;               /* success indicator                    */
MPARAM mpParam1;              /* Parameter 1 (select boolean)        */

case WM_SETSELECTION:
    /* if window is being selected, change focus to the window */
    if (SHORTFROMMP(mpParam1))
    {
        if ((hwndNewFocus =
            WinQueryFocus(HWND_DESKTOP)) != 0L)
            fSuccess = WinFocusChange(HWND_DESKTOP, hwndNewFocus,
                                      0L);
    }
```

WinFocusChange - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinFontDlg

WinFontDlg - Syntax

This dialog allows the user to select a font.

```
#define INCL_winstdfont
#include <os2.h>

HWND      hwndP; /* Parent-window handle. */
HWND      hwndO; /* Requested owner-window handle. */
PFONDLG   pfntd; /* Pointer to an initialized FONDLG structure. */
HWND      hwnd;  /* Font dialog window handle. */

hwnd = WinFontDlg(hwndP, hwndO, pfntd);
```

WinFontDlg Parameter - hwndP

hwndP ([HWND](#)) - input

Parent-window handle.

Parent-window handle of the created dialog window.

[HWND_DESKTOP](#)

The desktop window.

Other

Specified window.

WinFontDlg Parameter - hwndO

hwndO ([HWND](#)) - input

Requested owner-window handle.

Requested owner-window handle of the created dialog window.

The actual owner window is calculated using the algorithm specified in the description of the [WinLoadDlg](#) function.

WinFontDlg Parameter - pfntd

pfntd ([PFONTDLG](#)) - input
Pointer to an initialized [FONTDLG](#) structure.

WinFontDlg Return Value - hwnd

hwnd ([HWND](#)) - returns
Font dialog window handle.

If the flag is set by the application, the return value is the window handle of the font dialog, or NULLHANDLE if the dialog cannot be created. If the flag is not set, the return value is TRUE if dialog creation is successful, or NULLHANDLE if it is unsuccessful.

WinFontDlg - Parameters

hwndP ([HWND](#)) - input
Parent-window handle.

Parent-window handle of the created dialog window.

[HWND_DESKTOP](#)
The desktop window.
Other
Specified window.

hwndO ([HWND](#)) - input
Requested owner-window handle.

Requested owner-window handle of the created dialog window.

The actual owner window is calculated using the algorithm specified in the description of the [WinLoadDlg](#) function.

pfntd ([PFONTDLG](#)) - input
Pointer to an initialized [FONTDLG](#) structure.

hwnd ([HWND](#)) - returns
Font dialog window handle.

If the flag is set by the application, the return value is the window handle of the font dialog, or NULLHANDLE if the dialog cannot be created. If the flag is not set, the return value is TRUE if dialog creation is successful, or NULLHANDLE if it is unsuccessful.

WinFontDlg - Remarks

The *pfntd* parameter is required and the [FONTDLG](#) structure must be properly initialized.

Note: If the field in the [FONTDLG](#) structure, *pfntd* must be a pointer to a [FONTDLG](#) structure that is either static or allocated from the heap. This [FONTDLG](#) structure must not be allocated on the stack.

Upon return, the [FONTDLG](#) structure is updated with any user alterations and the field contains the value returned by the font dialog's [WinDismissDlg](#) function. By default this is the ID of the push button pressed to dismiss the dialog, DID_OK or DID_CANCEL, unless the application supplied additional push buttons in its template.

The pointer to the [FONTDLG](#) structure is placed in the QWL_USER field of the dialog's frame window. If in a custom font dialog procedure the pointer to the [FONTDLG](#) structure is desired, it should be queried from the frame window with [WinQueryWindowULong](#).

To subclass the default font dialog with a new template, the application must give the module and ID of the new font dialog template and the address of a dialog procedure for message handling. Window IDs in the range 0x0000 through 0x0FFF are reserved for the font dialog controls. IDs from outside this range must be chosen for any controls added to a custom font dialog.

When a modeless dialog is dismissed, the owner of the font dialog will receive a [WM_COMMAND](#) message with the parameter equal to the ID of the font dialog.

WinFontDlg - Example Code

This example displays a font selection dialog by using WinFontDlg, which allows the user to select a font.

```
#define INCL_WINSTDFONT      /* Window Standard Font Functions */
#include <os2.h>
#include <string.h>

HPS      hpsScreen;          /* Screen presentation space */
FONTDLG  pfdFontdlg;         /* Font dialog info structure */
HWND     hwndMain;          /* Window that owns the font dialog */
HWND     hwndFontDlg;        /* Font dialog window */

char  szFamilyname[FACESIZE];

/*****
/* Initially set all fields to 0.
*****/
memset(&pfdFontdlg,0,sizeof(FONTDLG));

/*****
/* Get handle to presentation space.
*****/
hpsScreen=WinGetPS(hwndMain);

/*****
/* Initialize those fields in the FONTDLG structure that are
/* used by the application
*****/
pfdFontdlg.cbSize=sizeof(FONTDLG);          /* Size of structure */
pfdFontdlg.hpsScreen=hpsScreen;             /* Screen presentation
/* space
szFamilyname[0]='\0';                       /* Use default font
pfdFontdlg.pszFamilyname=szFamilyname;      /* Provide buffer

pfdFontdlg.usFamilyBufLen = sizeof(szFamilyname);
/* Font family name
pfdFontdlg.fxPointSize=MAKEFIXED(10,0);    /* Font point size
pfdFontdlg.fl=FNTS_HELPBUTTON |
FNTS_CENTER;                              /* FNTS_* flags
pfdFontdlg.clrFore=CLR_BLACK;              /* Foreground color
pfdFontdlg.clrBack=CLR_WHITE;              /* Background color
pfdFontdlg.fAttrs.usCodePage=437;          /* Code page to select
/* from

/*****
/* Display the font dialog and get the font
*****/
hwndFontDlg=WinFontDlg(HWND_DESKTOP,hwndMain,&pfdFontdlg);
```

```

if(hwndFontDlg &&(pfdFontdlg.lReturn==DID_OK))
{
    /******
    /* Upon successful return of a font, the application can
    /* use font information selected by the user to create a
    /* font, load a font, and so forth
    /******
}

WinReleasePS(hpsScreen);

```

WinFontDlg - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

WinFreeErrorInfo

WinFreeErrorInfo - Syntax

This function releases memory allocated for an error-information block.

```

#define INCL_WINERRORS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

PERRINFO    perriErrorInfo; /* Error-information block whose memory is to be released. */
BOOL        rc;             /* Success indicator. */

rc = WinFreeErrorInfo(perriErrorInfo);

```

WinFreeErrorInfo Parameter - perriErrorInfo

perriErrorInfo ([PERRINFO](#)) - input
 Error-information block whose memory is to be released.

WinFreeErrorInfo Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE

Successful completion

FALSE

perriErrorInfo is not an error-information block for the current thread.

WinFreeErrorInfo - Parameters

perriErrorInfo ([PERRINFO](#)) - input
Error-information block whose memory is to be released.

rc ([BOOL](#)) - returns
Success indicator.

TRUE

Successful completion

FALSE

perriErrorInfo is not an error-information block for the current thread.

WinFreeErrorInfo - Related Functions

Related Functions

- [WinFreeErrorInfo](#)
- [WinGetErrorInfo](#)
- [WinGetLastError](#)

WinFreeErrorInfo - Example Code

This example frees memory allocated (by [WinGetErrorInfo](#)) for an error-information block using [WinFreeErrorInfo](#).

```
#define INCL_WINERRORS          /* Window Error Functions */
#include <os2.h>

BOOL fSuccess;                  /* success indicator */
ERRORID erridErrorCode; /* last error id code */
PERRINFO perriErrorInfo; /* error info structure */
HAB hab;                       /* anchor-block handle */

/* obtain error block and assign error code */
perriErrorInfo = WinGetErrorInfo(hab);
erridErrorCode = perriErrorInfo->idError;

/* free error block */
fSuccess = WinFreeErrorInfo(perriErrorInfo);
```

WinFreeErrorInfo - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinFreeFileDialogList

WinFreeFileDialogList - Syntax

This function frees the storage allocated by the file dialog when the dialog flag is set.

```
#define INCL_winstdfile
#include <os2.h>

PAPSZ    papszFQFilename; /* Pointer to a table of pointers of fully-qualified file names returned by the dia
BOOL     rc;              /* Success indicator. */

rc = WinFreeFileDialogList(papszFQFilename);
```

WinFreeFileDialogList Parameter - papszFQFilename

papszFQFilename ([PAPSZ](#)) - input

Pointer to a table of pointers of fully-qualified file names returned by the dialog.

WinFreeFileDialogList Return Value - rc

rc ([BOOL](#)) - returns

Success indicator.

TRUE

	Successful completion.
FALSE	Error occurred.

WinFreeFileDlgList - Parameters

papszFQFilename ([PAPSZ](#)) - input

Pointer to a table of pointers of fully-qualified file names returned by the dialog.

rc ([BOOL](#)) - returns

Success indicator.

TRUE

Successful completion.

FALSE

Error occurred.

WinFreeFileDlgList - Remarks

When the style flag is set and the user selects one or more files from the file name list box, the fully-qualified file names of the selected files are returned in the field of the [FILEDLG](#) structure. After the application retrieves all of the information it needs from the array, it should call WinFreeFileDlgList to free the storage.

WinFreeFileDlgList - Example Code

This example uses the WinFreeFileDlgList function to deallocate the table of file name pointers returned by the [WinFileDlg](#) function when the field of the [FILEDLG](#) structure.

```
#define INCL_WINSTDFILE /* Window Standard File Functions */
#include <os2.h>

BOOL fSuccess; /* Success indicator */
FILEDLG pfdFiledlg; /* File dialog info structure */
HWND hwndMain; /* Window that owns the file dialog */
HWND hwndDlg; /* File dialog window */

/*****
/* initialize FILEDLG structure
*****/
pfdFiledlg.cbSize = sizeof(FILEDLG); /* Size of structure */
pfdFiledlg.fl = FDS_MULTIPLESEL | FDS_HELPBUTTON | FDS_CENTER |
                FDS_OPEN_DIALOG; /* FDS_* flags

/*****
/* Set remaining fields here
*****/
.
.
.

/*****
/* Display the dialog and get the files
*****/

hwndDlg = WinFileDlg(HWND_DESKTOP, hwndMain, &pfdFiledlg);
```

```

if (hwndDlg && (pfdFileDlg.lReturn == DID_OK))
{
    /******
    /* Upon successful return of the files, open them for further */
    /* processing using the table of file name pointers          */
    /******

    /******
    /* Find out whether the pointer array was allocated          */
    /******

    if (pfdFileDlg.papszFQFilename)

        /******
        /* If so, free the table of file name pointers          */
        /******

        fSuccess = WinFreeFileDlgList(pfdFileDlg.papszFQFilename);
}

```

WinFreeFileDlgList - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

WinFreeFileIcon

WinFreeFileIcon - Syntax

This function frees an icon pointer that was originally allocated by [WinLoadFileIcon](#).

```

#define INCL_WINWORKPLACE
#include <os2.h>

HPOINTER    hptr; /* A pointer to an icon loaded by WinLoadFileIcon. */
BOOL        rc;   /* Success indicator. */

rc = WinFreeFileIcon(hptr);

```

WinFreeFileIcon Parameter - hptr

hptr ([HPOINTER](#)) - input
A pointer to an icon loaded by [WinLoadFileIcon](#).

WinFreeFileIcon Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinFreeFileIcon - Parameters

hptr ([HPOINTER](#)) - input
A pointer to an icon loaded by [WinLoadFileIcon](#).

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinFreeFileIcon - Related Functions

- Related Functions**
- [WinSetFileIcon](#)
 - [WinLoadFileIcon](#)

WinFreeFileIcon - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Related Functions](#)
- [Glossary](#)

WinGetClipPS

WinGetClipPS - Syntax

This method obtains a clipped cache presentation space.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND     hwnd;           /* Handle of window for which the presentation space is required. */
HWND     hwndClipWindow; /* Handle of window for clipping. */
ULONG    ulClipflags;    /* Clipping control flags. */
HPS      hps;            /* Presentation-space handle that can be used for drawing. */

hps = WinGetClipPS(hwnd, hwndClipWindow, ulClipflags);
```

WinGetClipPS Parameter - hwnd

hwnd (**HWND**) - input
Handle of window for which the presentation space is required.

WinGetClipPS Parameter - hwndClipWindow

hwndClipWindow (**HWND**) - input
Handle of window for clipping.

Values to be specified can be one of the following:

- HWND_BOTTOM**
Clip the last window in the sibling chain and continue clipping until the next window is *hwnd* or NULLHANDLE.
 - HWND_TOP**
Clip the first window in the sibling chain and continue clipping until the next window is *hwnd* or NULLHANDLE.
 - NULLHANDLE**
Clip all siblings to the window *hwnd*.
-

WinGetClipPS Parameter - ulClipflags

ulClipflags ([ULONG](#)) - input
Clipping control flags.

PSF_CLIPSIBLINGS
Clip out all siblings of *hwnd*.

PSF_CLIPCHILDREN
Clip out all children of *hwnd*.

PSF_CLIPUPWARDS
Taking *hwndClipWindow* as a reference window, clip out all sibling windows before *hwndClipWindow*. This value may not be used with PSF_CLIPDOWNWARDS.

PSF_CLIPDOWNWARDS
Taking *hwndClipWindow* as a reference window, clip out all sibling windows after *hwndClipWindow*. This value may not be used with PSF_CLIPUPWARDS.

PSF_LOCKWINDOWUPDATE
Calculate a presentation space that keeps a visible region even though output may be locked by the [WinLockWindowUpdate](#) function.

PSF_PARENTCLIP
Calculate a presentation space that uses the visible region of the parent of *hwnd* but with an origin calculated for *hwnd*.

WinGetClipPS Return Value - hps

hps ([HPS](#)) - returns
Presentation-space handle that can be used for drawing.

WinGetClipPS - Parameters

hwnd ([HWND](#)) - input
Handle of window for which the presentation space is required.

hwndClipWindow ([HWND](#)) - input
Handle of window for clipping.

Values to be specified can be one of the following:

HWND_BOTTOM
Clip the last window in the sibling chain and continue clipping until the next window is *hwnd* or NULLHANDLE.

HWND_TOP
Clip the first window in the sibling chain and continue clipping until the next window is *hwnd* or NULLHANDLE.

NULLHANDLE
Clip all siblings to the window *hwnd*.

ulClipflags ([ULONG](#)) - input
Clipping control flags.

PSF_CLIPSIBLINGS
Clip out all siblings of *hwnd*.

PSF_CLIPCHILDREN
Clip out all children of *hwnd*.

PSF_CLIPUPWARDS

Taking *hwndClipWindow* as a reference window, clip out all sibling windows before *hwndClipWindow*. This value may not be used with PSF_CLIPDOWNWARDS.

PSF_CLIPDOWNWARDS

Taking *hwndClipWindow* as a reference window, clip out all sibling windows after *hwndClipWindow*. This value may not be used with PSF_CLIPUPWARDS.

PSF_LOCKWINDOWUPDATE

Calculate a presentation space that keeps a visible region even though output may be locked by the [WinLockWindowUpdate](#) function.

PSF_PARENTCLIP

Calculate a presentation space that uses the visible region of the parent of *hwnd* but with an origin calculated for *hwnd*.

hps ([HPS](#)) - returns

Presentation-space handle that can be used for drawing.

WinGetClipPS - Remarks

The presentation space obtained by this method is a cache "micro-presentation space" present in the system. This can be used for simple drawing operations that do not depend on long-term data being stored in the presentation space.

When the application finishes using the clipped cache presentation space, it should destroy it using the [WinReleasePS](#) function.

WinGetClipPS - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)

An invalid window handle was specified.

WinGetClipPS - Related Functions

Related Functions

- [WinBeginPaint](#)
- [WinEnableWindowUpdate](#)
- [WinEndPaint](#)
- [WinExcludeUpdateRegion](#)
- [WinGetClipPS](#)
- [WinGetPS](#)
- [WinGetScreenPS](#)
- [WinInvalidateRect](#)
- [WinInvalidateRegion](#)
- [WinIsWindowShowing](#)
- [WinIsWindowVisible](#)
- [WinLockVisRegions](#)
- [WinOpenWindowDC](#)
- [WinQueryUpdateRect](#)
- [WinQueryUpdateRegion](#)
- [WinRealizePalette](#)
- [WinReleasePS](#)
- [WinShowWindow](#)

- [WinUpdateWindow](#)
- [WinValidateRect](#)
- [WinValidateRegion](#)

WinGetClipPS - Example Code

This example responds to an application defined message (IDM_FILL) and uses WinGetClipPS to obtain and associate a cached presentation space with a window, where the PS is clipped to the children of the window.

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#include <os2.h>

HWND    hwnd;                /* PS window */
HWND    hwndClip;            /* clipping window */
RECT    rcl;                 /* update region */
HPS     hps;                 /* presentation-space handle */

case IDM_FILL:
    hps = WinGetClipPS(hwnd,    /* handle of the PS window */
                       hwndClip, /* handle of clipping window */
                       PSF_CLIPCHILDREN); /* clipping flags */
    WinFillRect(hps, &rcl, CLR_WHITE);
    WinReleasePS(hps);
```

WinGetClipPS - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinGetCurrentTime

WinGetCurrentTime - Syntax

This function returns the current time.

```
#define INCL_WINTIMER /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB    hab;                /* Anchor-block handle. */
```

```
ULONG    ulTime; /* System-timer count. */  
  
ulTime = WinGetCurrentTime(hab);
```

WinGetCurrentTime Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinGetCurrentTime Return Value - ulTime

ulTime ([ULONG](#)) - returns
System-timer count.

The time is in milliseconds, from the system Initial Program Load (IPL). This is the same value as stored in the information segment.

WinGetCurrentTime - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

ulTime ([ULONG](#)) - returns
System-timer count.

The time is in milliseconds, from the system Initial Program Load (IPL). This is the same value as stored in the information segment.

WinGetCurrentTime - Related Functions

Related Functions

- [WinGetCurrentTime](#)
- [WinQueryMsgTime](#)
- [WinStartTimer](#)
- [WinStopTimer](#)

WinGetCurrentTime - Example Code

This example uses WinGetCurrentTime to return the current time.

```
#define INCL_WINTIMER          /* Window Timer Functions      */
#include <os2.h>

HAB  hab;          /* anchor-block handle      */
ULONG ulTime;      /* current time              */

ulTime = WinGetCurrentTime(hab);
```

WinGetCurrentTime - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinGetDlgMsg

WinGetDlgMsg - Syntax

This function obtains a message from the application's queue associated with the specified dialog.

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND  hwndDlg; /* Dialog-window handle. */
PQMSG pqmsg;   /* Message structure. */
BOOL  rc;      /* Continue message indicator. */

rc = WinGetDlgMsg(hwndDlg, pqmsg);
```

WinGetDlgMsg Parameter - hwndDlg

hwndDlg ([HWND](#)) - input
Dialog-window handle.

WinGetDlgMsg Parameter - pqmsg

pqmsg ([PQMSG](#)) - output
Message structure.

WinGetDlgMsg Return Value - rc

rc ([BOOL](#)) - returns
Continue message indicator.

TRUE
Message returned is not a [WM_QUIT](#) message and the dialog has not been dismissed.

FALSE
Message returned is a [WM_QUIT](#) message or the dialog has been dismissed.

WinGetDlgMsg - Parameters

hwndDlg ([HWND](#)) - input
Dialog-window handle.

pqmsg ([PQMSG](#)) - output
Message structure.

rc ([BOOL](#)) - returns
Continue message indicator.

TRUE
Message returned is not a [WM_QUIT](#) message and the dialog has not been dismissed.

FALSE
Message returned is a [WM_QUIT](#) message or the dialog has been dismissed.

WinGetDlgMsg - Remarks

This function enables a language that cannot support window procedures to provide the function of a modal dialog. The application creates a modeless dialog by the use of the [WinCreateDlg](#) or the [WinLoadDlg](#) functions and then issues this call to process messages only associated with the dialog.

The first time that this function is issued, the owner of the window specified by *hwndDlg* is disabled, thereby preventing input into windows other than the dialog. The owner of the window specified by *hwndDlg* is enabled when the [WinDismissDlg](#) function is issued either by the application or by the default dialog procedure.

If a [WM_QUIT](#) is encountered, WinGetDlgMsg itself issues a [WinDismissDlg](#) function, and posts the [WM_QUIT](#) message back to the queue so that the application main loop terminates in the normal way.

WinGetDlgMsg - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinGetDlgMsg - Related Functions

Related Functions

- [WinGetDlgMsg](#)
 - [WinBroadcastMsg](#)
 - [WinCreateDlg](#)
 - [WinCreateMsgQueue](#)
 - [WinDefDlgProc](#)
 - [WinDestroyMsgQueue](#)
 - [WinDismissDlg](#)
 - [WinDispatchMsg](#)
 - [WinDlgBox](#)
 - [WinGetMsg](#)
 - [WinInSendMessage](#)
 - [WinGetDlgMsg](#)
 - [WinLoadDlg](#)
 - [WinPeekMsg](#)
 - [WinPostMsg](#)
 - [WinPostQueueMsg](#)
 - [WinProcessDlg](#)
 - [WinGetDlgMsg](#)
 - [WinQueryMsgPos](#)
 - [WinQueryMsgTime](#)
 - [WinQueryQueueInfo](#)
 - [WinQueryQueueStatus](#)
 - [WinSendDlgItemMsg](#)
 - [WinSendMessage](#)
 - [WinSetClassMsgInterest](#)
 - [WinSetMsgInterest](#)
 - [WinSetMsgMode](#)
 - [WinSetSynchroMode](#)
 - [WinWaitMsg](#)
-

WinGetDlgMsg - Related Messages

Related Messages

- [WM_QUIT](#)
-

WinGetDlgMsg - Example Code

This example uses `WinGetDlgMsg` to provide a modal dialog. When the user causes an open message (application defined `IDM_OPEN`), the dialog is loaded and displayed; `WinGetDlgMsg` then loops, grabbing messages from the queue and calling `MyDlgRoutine`-the dialog procedure which processes the messages-with the appropriate parameters. When the dialog issues a `WM_QUIT`, `WinGetDlgMsg` returns `FALSE` and the loop ends, returning control to owner window.

```

#define INCL_WINDIALOGS          /* Window Dialog Mgr Functions */
#include <os2.h>

HWND    hwnd;          /* owner window */
HWND    hwndDlg;        /* dialog window */
PQMSG    pqmsgmsg;      /* message */

case IDM_OPEN:
    hwndDlg = WinLoadDlg(HWND_DESKTOP, /* parent is desk top */
                        hwnd,          /* owner window handle */
                        NULL,          /* modeless dialog */
                        0L,            /* load from .EXE */
                        DLG_ID,        /* dialog resource id */
                        NULL);         /* no dialog parameters */

    /* loop and process dialog messages until WM_QUIT, calling
       dialog procedure for each message */
    while (WinGetDlgMsg(hwndDlg, &qmsg))
        MyDlgRoutine(hwndDlg, qmsg.msg, qmsg.mp1, qmsg.mp2);
    break;

MRESULT MyDlgRoutine(HWND hwndDlg, ULONG usMsgid, MPARAM mp1,
                    MPARAM mp2)
{
    switch(usMsgid)
    {
        /*
        .
        . process messages
        .
        */

        default:
            return (WinDefDlgProc(hwndDlg, usMsgid, mp1, mp2));
    }
}

```

WinGetDlgMsg - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinGetErrorInfo

WinGetErrorInfo - Syntax

This function returns detailed error information.

```
#define INCL_WINERRORS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
PERRINFO perriErrorInfo; /* Error information. */

perriErrorInfo = WinGetErrorInfo(hab);
```

WinGetErrorInfo Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinGetErrorInfo Return Value - perriErrorInfo

perriErrorInfo ([PERRINFO](#)) - returns
Error information.

This structure contains information about the previous error code for the current thread:

NULL	No error information available
Other	Error information.

WinGetErrorInfo - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

perriErrorInfo ([PERRINFO](#)) - returns
Error information.

This structure contains information about the previous error code for the current thread:

NULL	No error information available
Other	Error information.

WinGetErrorInfo - Remarks

This function allocates a single private segment to contain the [ERRINFO](#) structure. All the pointers to string fields within the [ERRINFO](#) structure are offsets to memory within that segment.

The memory allocated by this function is not released until the returned pointer is passed to the [WinFreeErrorInfo](#) function.

WinGetErrorInfo - Related Functions

Related Functions

- [WinFreeErrorInfo](#)
- [WinGetErrorInfo](#)
- [WinGetLastError](#)

WinGetErrorInfo - Example Code

This example uses WinGetErrorInfo to obtain detailed error information, assigns the error code, and frees the error block with WinFreeErrorInfo.

```
#define INCL_WINERRORS          /* Window Error Functions      */
#include <os2.h>

BOOL fSuccess;                /* success indicator      */
ERRORID erridErrorCode; /* last error id code      */
PERRINFO perriErrorInfo; /* error info structure    */
HAB hab;                      /* anchor-block handle     */

/* obtain error block */
perriErrorInfo = WinGetErrorInfo(hab);
erridErrorCode = perriErrorInfo->idError;

/* free error block */
fSuccess = WinFreeErrorInfo(perriErrorInfo);
```

WinGetErrorInfo - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinGetKeyState

WinGetKeyState - Syntax

This function returns the state of the key at the time that the last message obtained from the queue was posted.

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndDesktop; /* Desktop-window handle. */
LONG    vkey;         /* Virtual key value. */
LONG    lKeyState;    /* Key state. */

lKeyState = WinGetKeyState(hwndDesktop, vkey);
```

WinGetKeyState Parameter - hwndDesktop

hwndDesktop (**HWND**) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

WinGetKeyState Parameter - vkey

vkey (**LONG**) - input
Virtual key value.

Contains the virtual key value in the low-order byte, and zero in the high-order byte. See remarks for possible values.

WinGetKeyState Return Value - lKeyState

lKeyState (**LONG**) - returns
Key state.

This value is the OR combination of the following bits:

0x0001	The key has been pressed an odd number of times since the system has been started.
0x8000	The key is down.

WinGetKeyState - Parameters

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP
The desktop-window handle
Other
Specified desktop-window handle.

key ([LONG](#)) - input
Virtual key value.

Contains the virtual key value in the low-order byte, and zero in the high-order byte. See remarks for possible values.

lKeyState ([LONG](#)) - returns
Key state.

This value is the OR combination of the following bits:

0x0001
The key has been pressed an odd number of times since the system has been started.
0x8000
The key is down.

WinGetKeyState - Remarks

See also the [WinGetPhysKeyState](#) function. This function is used to determine whether a virtual key is up, down, or toggled.

This function can be used to obtain the state of the pointing device buttons with the VK_BUTTON1, VK_BUTTON2, and VK_BUTTON3 virtual key codes. The following are the possible values for the *key* parameter:

VK_BUTTON1
VK_BUTTON2
VK_BUTTON3

VK_ALT
VK_ALTF
VK_CTRL
VK_SHIFT
VK_SPACE
VK_BACKSPACE
VK_TAB
VK_BACKTAB

VK_BREAK
VK_PAUSE
VK_NEWLINE

VK_END
VK_HOME
VK_LEFT
VK_UP
VK_DOWN
VK_RIGHT
VK_PAGEDOWN
VK_PAGEUP
VK_DELETE

VK_INSERT

VK_CAPSLOCK
VK_NUMLOCK
VK_SCROLLLOCK
VK_PRINTSCRN
VK_ENTER
VK_ESC
VK_SYSRO

VK_ENDDRAG
VK_CLEAR
VK_EREOF
VK_PA1
VK_ATTN
VK_CRSEL
VK_EXCEL
VK_COPY
VK_BLK1
VK_BLK2
VK_MENU

VK_BIDI_FIRST
VK_BIDI_LAST
VK_DBCSFIRST
VK_DBCSLAST
VK_USERFIRST
VK_USERLAST

VK_F1
VK_F2
VK_F3
VK_F4
VK_F5
VK_F6
VK_F7
VK_F8
VK_F9
VK_F10
VK_F11
VK_F12
VK_F13
VK_F14
VK_F15
VK_F16
VK_F17
VK_F18
VK_F19
VK_F20
VK_F21
VK_F22
VK_F23
VK_F24

WinGetKeyState - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinGetKeyState - Related Functions

Related Functions

- [WinEnablePhysInput](#)
- [WinFocusChange](#)
- [WinGetKeyState](#)
- [WinGetPhysKeyState](#)
- [WinQueryFocus](#)
- [WinSetFocus](#)
- [WinSetKeyboardStateTable](#)

WinGetKeyState - Example Code

This example uses WinGetKeyState to check if mouse button 1 was depressed when a WM_TIMER message was received. A high pitched beep is emitted if it was depressed, and a low pitched beep if it was not.

```
#define INCL_WININPUT      /* Window Input functions */
#define INCL_DOSPROCESS    /* OS/2 Process functions */
#include <os2.h>

LONG   lKeyState;          /* Key state */
LONG   vkey;               /* Virtual key value */

case WM_TIMER:
    /* Get the key state of mouse button 1 */
    vkey = VK_BUTTON1;
    lKeyState = WinGetKeyState(HWND_DESKTOP, vkey);

    /* Emit a high pitched beep if mouse button 1 is depressed */
    /* when the timer message occurred; otherwise, emit a low */
    /* pitched beep */
    if (lKeyState & 0x8000)
        DosBeep(1000,100L);
    else
        DosBeep(200,100L);
```

WinGetKeyState - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinGetLastError

WinGetLastError - Syntax

This function returns the error code set by the failure of a Presentation Manager function.

```
#define INCL_WINERRORS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
ERRORID  erridErrorCode; /* Last error code. */

erridErrorCode = WinGetLastError(hab);
```

WinGetLastError Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinGetLastError Return Value - erridErrorCode

erridErrorCode ([ERRORID](#)) - returns
Last error code.

The returned error code is a 32-bit quantity. The high order 16 bits is a severity code. The low order 16 bits is the error code.

See the Notes for more information on using the returned ERRORID value.

WinGetLastError - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

erridErrorCode ([ERRORID](#)) - returns
Last error code.

The returned error code is a 32-bit quantity. The high order 16 bits is a severity code. The low order 16 bits is the error code.

See the Notes for more information on using the returned ERRORID value.

WinGetLastError - Remarks

Returns the last nonzero error code, and sets the error code to zero.

The current error code is reset to zero.

In multiple thread applications where there are multiple anchor blocks, errors are stored in the anchor block created by the [WinInitialize](#) function of the thread invoking a call. The last error for the process and thread on which this function call is made will be returned.

The returned error code is a 32-bit quantity. The high order 16 bits is a severity code. The low order 16 bits is the error code.

The severity codes are defined as follows:

```
#define SEVERITY_NOERROR          0x0000
#define SEVERITY_WARNING         0x0004
#define SEVERITY_ERROR           0x0008
#define SEVERITY_SEVERE          0x000C
#define SEVERITY_UNRECOVERABLE   0x0010
```

Error codes are described in

Two macros have been defined to simplify extracting severity codes and error codes from the returned ERRORID value.

```
/* Extract severity from an errorid */
#define ERRORIDSEV(errid)          (HIUSHORT(errid))

/* Extract error number from an errorid */
#define ERRORIDERROR(errid)       (LOUSHORT(errid))
```

WinGetLastError - Related Functions

Related Functions

- [WinFreeErrorInfo](#)
- [WinGetErrorInfo](#)
- WinGetLastError

WinGetLastError - Example Code

This example uses WinGetLastError to obtain the error code corresponding to the last nonzero error for the specified anchor block. If only the error code is required, this function is preferable to the WinGetErrorInfo/WinFreeErrorInfo call sequence.

```
#define INCL_WINERRORS          /* Window Error Functions */
#include <os2.h>

ERRORID  erridErrorCode; /* last error id code */
HAB     hab;            /* anchor-block handle */

/* get last nonzero error for this anchor block */
erridErrorCode = WinGetLastError(hab);
```

WinGetLastError - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinGetMaxPosition

WinGetMaxPosition - Syntax

The WinGetMaxPosition function fills an SWP structure with the maximized-window size and position.

```
#define INCL_WINFRAMEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd;          /* Frame-window handle. */
PSWP    pswp;          /* Set window position structure. */
BOOL    fSuccess;      /* Success indicator. */

fSuccess = WinGetMaxPosition(hwnd, pswp);
```

WinGetMaxPosition Parameter - hwnd

hwnd (**HWND**) - input
Frame-window handle.

Identifies the window whose maximum size will be retrieved.

WinGetMaxPosition Parameter - pswp

pswp (**PSWP**) - output
Set window position structure.

Points to the SWP structure that retrieves the size and position of a maximized window.

The SWP_SIZE and SWP_MOVE indicators are set in this parameter on return from this call, implying that the parameters have been initialized.

WinGetMaxPosition Return Value - fSuccess

fSuccess (BOOL) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

WinGetMaxPosition - Parameters

hwnd (HWND) - input
Frame-window handle.

Identifies the window whose maximum size will be retrieved.

pswp (PSWP) - output
Set window position structure.

Points to the SWP structure that retrieves the size and position of a maximized window.

The SWP_SIZE and SWP_MOVE indicators are set in this parameter on return from this call, implying that the parameters have been initialized.

fSuccess (BOOL) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

WinGetMaxPosition - Example Code

This example uses WinGetMaxPosition to determine the maximized position for the window in response to a maximize message (WM_MINMAXFRAME), and then calls WinSetWindowPos to maximize the window to that position.

```
#define INCL_WINFRAMEMGR          /* Window Frame Functions */
#include <os2.h>

BOOL fSuccess;                  /* Success indicator */
HWND hwnd;                     /* window handle */
MPARAM mpParam1;               /* Parameter 1 (window position) */
PSWP pSwp;                     /* Set window position structure */

case WM_MINMAXFRAME:
    pSwp = (PSWP)PVOIDFROMMP(mpParam1);

    switch(pSwp->fl)
    {
        case SWP_MAXIMIZE:
            fSuccess = WinGetMaxPosition(hwnd, pSwp);
```

```

        WinSetWindowPos(hwnd, 0L,
        pSwp->x,          /* x pos */
        pSwp->y,          /* y pos */
        pSwp->cx,         /* x size */
        pSwp->cy,         /* y size */
        SWP_MAXIMIZE);   /* flags */
        break;
    }

```

WinGetMaxPosition - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Example Code](#)

[Glossary](#)

WinGetMinPosition

WinGetMinPosition - Syntax

This function returns the position to which a window is minimized.

```

#define INCL_WINFRAMEMEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwnd; /* Frame-window handle. */
PSWP      pswp; /* Set window position structure. */
PPOINTL   pptl; /* Preferred position. */
BOOL      rc;   /* Success indicator. */

rc = WinGetMinPosition(hwnd, pswp, pptl);

```

WinGetMinPosition Parameter - hwnd

hwnd (**HWND**) - input
Frame-window handle.

WinGetMinPosition Parameter - pswp

pswp (PSWP) - output
Set window position structure.

The SWP_SIZE and SWP_MOVE indicators are set in this parameter on return from this function, implying that the parameters have been initialized.

WinGetMinPosition Parameter - pptl

pptl (PPOINTL) - input
Preferred position.

- NULL
System is to choose the position
- Other
System is to choose the position nearest to the specified point.

WinGetMinPosition Return Value - rc

rc (BOOL) - returns
Success indicator.

- TRUE
Successful completion.

The WS_MINIMIZE style is set for *hwnd*. This enables the system to determine which other frame windows are minimized, during the enumeration process performed by this function.

Also, the window words QWS_XMINIMIZE and QWS_YMINIMIZE for *hwnd* are initialized. This enables the system to ensure that no windows that have been, or are being, minimized use the same position.
- FALSE
Error occurred.

WinGetMinPosition - Parameters

hwnd (HWND) - input
Frame-window handle.

pswp (PSWP) - output
Set window position structure.

The SWP_SIZE and SWP_MOVE indicators are set in this parameter on return from this function, implying that the parameters have been initialized.

pptl (PPOINTL) - input
Preferred position.

NULL

Other	System is to choose the position
	System is to choose the position nearest to the specified point.
rc (BOOL) - returns Success indicator.	
TRUE	Successful completion.
	The WS_MINIMIZE style is set for <i>hwnd</i> . This enables the system to determine which other frame windows are minimized, during the enumeration process performed by this function.
	Also, the window words QWS_XMINIMIZE and QWS_YMINIMIZE for <i>hwnd</i> are initialized. This enables the system to ensure that no windows that have been, or are being, minimized use the same position.
FALSE	Error occurred.

WinGetMinPosition - Remarks

This function chooses the position for a minimized window. It enumerates all the siblings of the specified window to determine the first available position.

WinGetMinPosition - Related Functions

Related Functions

- [WinGetMinPosition](#)
- [WinQueryActiveWindow](#)
- [WinQueryWindowPos](#)
- [WinSaveWindowPos](#)
- [WinSetActiveWindow](#)
- [WinSetMultWindowPos](#)
- [WinSetWindowPos](#)

WinGetMinPosition - Example Code

This example uses WinGetMinPosition to determine the minimized position for the window in response to a minimize message (WM_MINMAXFRAME), and then calls WinSetWindowPos to minimize the window to that position.

```
#define INCL_WINFRAMEMGR          /* Window Frame Functions */
#include <os2.h>

BOOL fSuccess;                  /* Success indicator */
HWND hwnd;                     /* window handle */
MPARAM mpParam1;               /* Parameter 1 (window position) */
PSWP pSwp;                    /* Set window position structure */

case WM_MINMAXFRAME:
    pSwp = (PSWP)PVOIDFROMMP(mpParam1);

    switch(pSwp->fl)
    {
        case SWP_MINIMIZE:
            fSuccess = WinGetMinPosition(hwnd, pSwp, NULL);
```

```

        WinSetWindowPos(hwnd, 0L,
            pSwp->x,          /* x pos */
            pSwp->y,          /* y pos */
            pSwp->cx,         /* x size */
            pSwp->cy,         /* y size */
            SWP_MINIMIZE);   /* flags */
    break;
}

```

WinGetMinPosition - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinGetMsg

WinGetMsg - Syntax

This function gets, waiting if necessary, a message from the thread's message queue and returns when a message conforming to the filtering criteria is available.

```

#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
PQMSG    pqmsgmsg;     /* Message structure. */
HWND     hwndFilter;   /* Window filter. */
ULONG    ulFirst;      /* First message identity. */
ULONG    ulLast;       /* Last message identity. */
BOOL     rc;           /* Continue message indicator. */

rc = WinGetMsg(hab, pqmsgmsg, hwndFilter,
    ulFirst, ulLast);

```

WinGetMsg Parameter - hab

hab ([HAB](#)) - input

Anchor-block handle.

WinGetMsg Parameter - pqmsgmsg

pqmsgmsg ([PQMSG](#)) - output
Message structure.

WinGetMsg Parameter - hwndFilter

hwndFilter ([HWND](#)) - input
Window filter.

WinGetMsg Parameter - ulFirst

ulFirst ([ULONG](#)) - input
First message identity.

WinGetMsg Parameter - ulLast

ulLast ([ULONG](#)) - input
Last message identity.

WinGetMsg Return Value - rc

rc ([BOOL](#)) - returns
Continue message indicator.

TRUE

Message returned is not a [WM_QUIT](#) message

FALSE

Message returned is a [WM_QUIT](#) message.

WinGetMsg - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

pqmsgmsg ([PQMSG](#)) - output
Message structure.

hwndFilter ([HWND](#)) - input
Window filter.

ulFirst ([ULONG](#)) - input
First message identity.

ulLast ([ULONG](#)) - input
Last message identity.

rc ([BOOL](#)) - returns
Continue message indicator.

TRUE	Message returned is not a WM_QUIT message
FALSE	Message returned is a WM_QUIT message.

WinGetMsg - Remarks

If system or queue hooks are installed, they are called before this function returns.

rc is generally used to determine when to terminate the application's main loop and exit the program.

hwndFilter constrains the returned message to be for a specific window or its children. When *hwndFilter* is null, the returned message can be for any window. The message identity is restricted to the range of message identities specified by *ulFirst* and *ulLast* inclusive. When *ulFirst* and *ulLast* are both zero, any message satisfies the range constraint. When *ulFirst* is greater than *ulLast*, messages except those whose identities lie between *ulFirst* and *ulLast* are eligible to be returned. Messages that do not conform to the filtering criteria remain in the queue.

When *hwndFilter* is null, and *ulFirst* and *ulLast* are both zero, all messages are returned in the order that they were posted to the queue.

By using filtering, messages can be processed in an order that is different from the one in the queue. Filtering is used in situations where applications receive messages of a particular type, rather than having to deal with other types of message at an inconvenient point in the logic of the application. For example, when a "mouse down" message is received, filtering can be used to wait for the "mouse up" message without having to be concerned with receiving other messages.

These constants can also be used when filtering messages:

WM_MOUSEFIRST	Lowest value pointing device message
WM_MOUSELAST	Highest value pointing device message
WM_BUTTONCLICKFIRST	Lowest value pointing device button click message
WM_BUTTONCLICKLAST	Highest value pointing device button click message
WM_DDE_FIRST	Lowest value DDE message
WM_DDE_LAST	Highest value DDE message.

Great care must be taken if filtering is used, to ensure that a message that satisfies the specification of the filtering parameters can occur, otherwise this function cannot complete. For example, calling this function with *ulFirst* and *ulLast* equal to [WM_CHAR](#) and with *hwndFilter* set to a window handle that does not have the input focus, prevents this function from returning.

Keystrokes are passed to the [WinTranslateAccel](#) call, which implies that accelerator keys are translated into [WM_COMMAND](#) or [WM_SYSCOMMAND](#) messages, and so are not seen as [WM_CHAR](#) messages by the application.

Note: An application must be prepared to receive messages other than those documented in this publication. All messages that an application does not want to handle should be dispatched to the appropriate window procedure using the [WinDispatchMsg](#) function.

WinGetMsg - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinGetMsg - Related Functions

Related Functions

- [WinBroadcastMsg](#)
 - [WinCancelShutdown](#)
 - [WinCreateMsgQueue](#)
 - [WinDestroyMsgQueue](#)
 - [WinDispatchMsg](#)
 - [WinGetDlgMsg](#)
 - [WinGetMsg](#)
 - [WinInSendMessage](#)
 - [WinPeekMsg](#)
 - [WinPostMsg](#)
 - [WinPostQueueMsg](#)
 - [WinQueryMsgPos](#)
 - [WinQueryMsgTime](#)
 - [WinQueryQueueInfo](#)
 - [WinQueryQueueStatus](#)
 - [WinSendDlgItemMsg](#)
 - [WinSendMessage](#)
 - [WinSetClassMsgInterest](#)
 - [WinSetMsgInterest](#)
 - [WinSetMsgMode](#)
 - [WinSetSynchroMode](#)
 - [WinWaitMsg](#)
-

WinGetMsg - Related Messages

Related Messages

- [WM_CHAR](#)
 - [WM_QUIT](#)
 - [WM_SYSCOMMAND](#)
-

WinGetMsg - Example Code

This example uses WinGetMsg to continually loop and retrieve messages from the message queue until a WM_QUIT message occurs.

```
#define INCL_WINMESSAGEGR      /* Window Message Functions */
#define INCL_WINWINDOWMGR     /* Window Manager Functions */
#include <os2.h>
```

```

HAB      hab;           /* anchor-block handle          */
HMQ      hmq;           /* message queue handle      */
PQMSG    pqmsgmsg;      /* message                    */

hab = WinInitialize(0);          /* initialize PM */

hmq = WinCreateMsgQueue(hab, 0); /* create default size queue */

/*
.
.  initialize windows
.
*/

/* get and dispatch messages from queue */
while (WinGetMsg(hab, &qmsgmsg, 0, 0, 0))
    WinDispatchMsg(hab, &qmsgmsg);

```

WinGetMsg - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinGetNextWindow

WinGetNextWindow - Syntax

This function gets the window handle of the next window in a specified enumeration list.

```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HENUM    henum;          /* Enumeration handle. */
HWND     hwndNext;       /* Next window handle in enumeration list. */

hwndNext = WinGetNextWindow(henum);

```

WinGetNextWindow Parameter - henum

henum ([HENUM](#)) - input
Enumeration handle.

Returned by previous call to the [WinBeginEnumWindows](#) call.

WinGetNextWindow Return Value - hwndNext

hwndNext ([HWND](#)) - returns
Next window handle in enumeration list.

NULLHANDLE	Error occurred, <i>henum</i> was invalid, or all the windows have been enumerated.
Other	Next window handle.

WinGetNextWindow - Parameters

henum ([HENUM](#)) - input
Enumeration handle.

Returned by previous call to the [WinBeginEnumWindows](#) call.

hwndNext ([HWND](#)) - returns
Next window handle in enumeration list.

NULLHANDLE	Error occurred, <i>henum</i> was invalid, or all the windows have been enumerated.
Other	Next window handle.

WinGetNextWindow - Remarks

Enumeration starts with the topmost child window and then proceeds downward through the enumeration list, in z-order at the time the [WinBeginEnumWindows](#) was issued, until all the windows have been enumerated. At this point, the call returns `NULLHANDLE`. The enumeration then wraps and the handle of the topmost child window is returned on the next call. This function does not lock windows.

WinGetNextWindow - Errors

Possible returns from [WinGetLastError](#)

`PMERR_INVALID_HENUM` (0x101C)
An invalid enumeration handle was specified.

WinGetNextWindow - Related Functions

Related Functions

- [WinBeginEnumWindows](#)
 - [WinEndEnumWindows](#)
 - [WinEnumDlgItem](#)
 - [WinGetNextWindow](#)
 - [WinIsChild](#)
 - [WinMultWindowFromIDs](#)
 - [WinQueryWindow](#)
 - [WinSetOwner](#)
 - [WinSetParent](#)
-

WinGetNextWindow - Example Code

This example moves through all the child windows in a enumeration list, using an enumeration handle provided by WinBeginEnumWindows; for each child window, the class name is queried and placed in a buffer.

```
#define INCL_WINWINDOWMGR          /* Window Manager Functions */
#include <os2.h>

HWND  hwndParent;          /* Handle of the window whose child windows
                           are to be enumerated */
HWND  hwndNext;            /* current enumeration handle */
HENUM  henum;              /* enumeration handle */
BOOL   fSuccess;           /* success indicator */
SHORT  sRetLen;            /* returned string length */
SHORT  sLength = 10;       /* string buffer length */
char   pchBuffer[10];      /* string buffer */

hwndParent = HWND_DESKTOP;

henum = WinBeginEnumWindows(hwndParent);

while ((hwndNext = WinGetNextWindow(henum)) != NULLHANDLE)
    sRetLen = WinQueryClassName(hwndNext, sLength, pchBuffer);

fSuccess = WinEndEnumWindows (henum);
```

WinGetNextWindow - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinGetPhysKeyState

WinGetPhysKeyState - Syntax

This function returns the physical key state.

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndDeskTop; /* Desktop-window handle. */
LONG    sc;           /* Hardware scan code. */
LONG    lKeyState;    /* Key state. */

lKeyState = WinGetPhysKeyState(hwndDeskTop,
                               sc);
```

WinGetPhysKeyState Parameter - hwndDeskTop

hwndDeskTop (**HWND**) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

WinGetPhysKeyState Parameter - sc

sc (**LONG**) - input
Hardware scan code.

Contains the scan code value in the low-order byte, and zero in the high-order byte.

WinGetPhysKeyState Return Value - lKeyState

lKeyState (**LONG**) - returns
Key state.

This value is the OR combination of the following bits:

0x0001	The key has been pressed since the last time this function was issued, or since the system has been started if this is the first time the call has been issued.
0x8000	The key is down.

WinGetPhysKeyState - Parameters

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

sc ([LONG](#)) - input
Hardware scan code.

Contains the scan code value in the low-order byte, and zero in the high-order byte.

lKeyState ([LONG](#)) - returns
Key state.

This value is the OR combination of the following bits:

0x0001	The key has been pressed since the last time this function was issued, or since the system has been started if this is the first time the call has been issued.
0x8000	The key is down.

WinGetPhysKeyState - Remarks

This function returns information about the asynchronous (interrupt level) state of the virtual key indicated by the *sc* parameter.

This function returns the physical state of the key; it is not synchronized to the processing of input (see the [WinGetKeyState](#) function).

WinGetPhysKeyState - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinGetPhysKeyState - Related Functions

Related Functions

- [WinEnablePhysInput](#)
- [WinFocusChange](#)
- [WinGetKeyState](#)
- [WinGetPhysKeyState](#)
- [WinQueryFocus](#)
- [WinSetFocus](#)
- [WinSetKeyboardStateTable](#)

WinGetPhysKeyState - Example Code

This example uses `WinGetPhysKeyState` to check the current state of the caps lock key; if it is depressed, a high pitch beep is emitted, while a low pitch beep is emitted if it is not depressed.

```
#define INCL_WININPUT      /* Window Input functions */
#define INCL_DOSPROCESS    /* OS/2 Process functions */
#include <os2.h>

LONG   lKeyState;          /* Key state          */
LONG   lScancode;         /* Scan code value   */

/* Get the physical key state for the caps lock key */
lScancode = 0x3a;
lKeyState = WinGetPhysKeyState(HWND_DESKTOP, lScancode);

/* Emit the high pitched beep if the caps lock is currently depressed; */
/* otherwise, emit a low pitched beep                                */
if (lKeyState & 0x8000)
    DosBeep(1000,100L);
else
    DosBeep(200,100L);
```

WinGetPhysKeyState - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinGetPS

WinGetPS - Syntax

This method gets a cache presentation space.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND    hwnd; /* Handle of window for which the presentation space is required. */
HPS     hps; /* Presentation-space handle that can be used for drawing in the window. */

hps = WinGetPS(hwnd);
```

WinGetPS Parameter - hwnd

- hwnd (HWND)** - input
Handle of window for which the presentation space is required.
 - HWND_DESKTOP
The desktop-window handle; a presentation space for the whole of the desktop window is returned
 - Other
Handle of window for which the presentation space is required.

WinGetPS Return Value - hps

- hps (HPS)** - returns
Presentation-space handle that can be used for drawing in the window.

WinGetPS - Parameters

- hwnd (HWND)** - input
Handle of window for which the presentation space is required.
 - HWND_DESKTOP
The desktop-window handle; a presentation space for the whole of the desktop window is returned
 - Other
Handle of window for which the presentation space is required.
- hps (HPS)** - returns
Presentation-space handle that can be used for drawing in the window.

WinGetPS - Remarks

The presentation space created by this method is a cache "micro presentation space" present in the system. This can be used for simple drawing operations that do not depend on long-term data being stored in the presentation space.

The initial state of the presentation space is the same as that of a presentation space created using [GpiCreatePS](#). The color table is in default color index mode. The visible region associated with *hps* depends upon the window and class styles of *hwnd*:

Style	Visible region of presentation space
WS_CLIPCHILDREN	All child windows of the window are excluded.
WS_CLIPSIBLINGS	All the sibling windows of <i>hwnd</i> are excluded.
CS_PARENTCLIP	Is the same as that of the parent window of the window.
	The presentation space origin is established normally, that is, relative to the lower left of the window itself, not its parent.
	This style optimizes the use of the presentation space cache by minimizing the calculation of the visible region for child windows.

Any presentation space created by [WinGetPS](#) must be released by calling [WinReleasePS](#). This should be done before the application terminates.

Note: This call requires the presence of a message queue and should not be made until after the application's message queue has been created.

WinGetPS - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinGetPS - Related Functions

Related Functions

- [WinBeginPaint](#)
- [WinEnableWindowUpdate](#)
- [WinEndPaint](#)
- [WinExcludeUpdateRegion](#)
- [WinGetClipPS](#)
- [WinGetPS](#)
- [WinGetScreenPS](#)
- [WinInvalidateRect](#)
- [WinInvalidateRegion](#)
- [WinIsWindowShowing](#)
- [WinIsWindowVisible](#)
- [WinLockVisRegions](#)
- [WinOpenWindowDC](#)
- [WinQueryUpdateRect](#)
- [WinQueryUpdateRegion](#)
- [WinRealizePalette](#)
- [WinReleasePS](#)
- [WinShowWindow](#)
- [WinUpdateWindow](#)
- [WinValidateRect](#)

- [WinValidateRegion](#)

WinGetPS - Example Code

This example processes an application-defined message (IDM_FILL). It calls WinGetPS to get a presentation space to the entire window. It gets the dimensions of the current window, fills the window, and calls WinReleasePS to release the presentation space.

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#include <os2.h>

HWND    hwnd;                /* parent window */
RECTL   rcl;                 /* update region */
HPS     hps;                 /* presentation-space handle */

case IDM_FILL:
    hps = WinGetPS(hwnd);     /* get presentation space for */
                               /* the entire window */

    WinQueryWindowRect(hwnd, &rcl); /* get window dimensions */

    WinFillRect(hps, &rcl, CLR_WHITE); /* clear entire window */

    WinReleasePS(hps);         /* release the presentation */
                               /* space */

    return 0L;
```

WinGetPS - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinGetScreenPS

WinGetScreenPS - Syntax

This method returns a presentation space that can be used for drawing anywhere on the screen.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
```

```

HWND    hwndDesktop; /* Desktop-window handle. */
HPS     hpsScreenPS; /* Presentation-space handle. */

hpsScreenPS = WinGetScreenPS(hwndDesktop);

```

WinGetScreenPS Parameter - hwndDesktop

hwndDesktop (**HWND**) - input
Desktop-window handle.

HWND_DESKTOP
The desktop-window handle

Other
Specified desktop-window handle.

WinGetScreenPS Return Value - hpsScreenPS

hpsScreenPS (**HPS**) - returns
Presentation-space handle.

A micro presentation space that can be used for drawing over the entire desktop window (the whole screen).

NULLHANDLE
hwndDesktop is not **HWND_DESKTOP** or a desktop window handle obtained from the [WinQueryDesktopWindow](#) function.

Other
Presentation space handle.

WinGetScreenPS - Parameters

hwndDesktop (**HWND**) - input
Desktop-window handle.

HWND_DESKTOP
The desktop-window handle

Other
Specified desktop-window handle.

hpsScreenPS (**HPS**) - returns
Presentation-space handle.

A micro presentation space that can be used for drawing over the entire desktop window (the whole screen).

NULLHANDLE
hwndDesktop is not **HWND_DESKTOP** or a desktop window handle obtained from the [WinQueryDesktopWindow](#) function.

Other

Presentation space handle.

WinGetScreenPS - Remarks

Take great care when using this method. The returned presentation space is not clipped to any of the other windows present on the screen. Thus it is possible to draw in regions belonging to windows of other threads and processes.

The [WinLockWindowUpdate](#) function should be used to avoid simultaneous updates to the same part of the screen. This does not cause the presentation space returned by this function to become clipped in any way. Care of the appearance of windows of other threads is still the responsibility of the user of the screen presentation space.

When the application finishes using the screen presentation space, it should be destroyed using the [WinReleasePS](#) call.

WinGetScreenPS - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinGetScreenPS - Related Functions

Related Functions

- [WinBeginPaint](#)
 - [WinEnableWindowUpdate](#)
 - [WinEndPaint](#)
 - [WinExcludeUpdateRegion](#)
 - [WinGetClipPS](#)
 - [WinGetPS](#)
 - [WinGetScreenPS](#)
 - [WinInvalidateRect](#)
 - [WinInvalidateRegion](#)
 - [WinIsWindowShowing](#)
 - [WinIsWindowVisible](#)
 - [WinLockVisRegions](#)
 - [WinOpenWindowDC](#)
 - [WinQueryUpdateRect](#)
 - [WinQueryUpdateRegion](#)
 - [WinRealizePalette](#)
 - [WinReleasePS](#)
 - [WinShowWindow](#)
 - [WinUpdateWindow](#)
 - [WinValidateRect](#)
 - [WinValidateRegion](#)
-

WinGetScreenPS - Example Code

This example processes an application-defined message (IDM_FILL). It calls WinGetScreenPS to get a presentation space for the entire

desktop window, gets the dimensions of the current window, fills the window, and calls WinReleasePS to release the presentation space.

```
#define INCL_WINWINDOWMGR          /* Window Manager Functions */
#include <os2.h>

HWND    hwnd;          /* parent window */
RECT    rcl;           /* update region */
HPS     hps;           /* presentation-space handle */

case IDM_FILL:
    /* get presentation space for the entire desktop */
    hps = WinGetScreenPS(HWND_DESKTOP);

    WinQueryWindowRect(hwnd, &rcl);    /* get window dimensions */

    WinFillRect(hps, &rcl, CLR_WHITE); /* clear entire window */

    WinReleasePS(hps);                /* release the presentation space */
    return 0L;
```

WinGetScreenPS - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinGetSysBitmap

WinGetSysBitmap - Syntax

This function returns a handle to one of the standard bit maps provided by the system.

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndDesktop; /* Desktop-window handle. */
ULONG   ibm;         /* System bit-map index value. */
HBITMAP hbm;         /* System bit-map handle. */

hbm = WinGetSysBitmap(hwndDesktop, ibm);
```

WinGetSysBitmap Parameter - hwndDesktop

hWndDesktop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP
The desktop-window handle
Other
Specified desktop-window handle.

WinGetSysBitmap Parameter - ibm

ibm ([ULONG](#)) - input
System bit-map index value.

SBMP_SYSMENU
System menu
SBMP_SYSMENUDEP
System menu in depressed state
SBMP_SBUPARROW
Scroll bar up arrow
SBMP_SBUPARROWDEP
Scroll bar up arrow in depressed state
SBMP_SBUPARROWDIS
Scroll bar up arrow in disabled state
SBMP_SBDNARROW
Scroll bar down arrow
SBMP_SBDNARROWDEP
Scroll bar down arrow in depressed state
SBMP_SBDNARROWDIS
Scroll bar down arrow in disabled state
SBMP_SBRGARROW
Scroll bar right arrow
SBMP_SBRGARROWDEP
Scroll bar right arrow in depressed state
SBMP_SBRGARROWDIS
Scroll bar right arrow in disabled state
SBMP_SBLFARROW
Scroll bar left arrow
SBMP_SBLFARROWDEP
Scroll bar left arrow in depressed state
SBMP_SBLFARROWDIS
Scroll bar left arrow in disabled state
SBMP_MENUCHECK
Menu check mark
SBMP_MENUATTACHED
Cascading menu mark
SBMP_CHECKBOXES
Check box or radio button check marks
SBMP_COMBODOWN
Combobox down arrow
SBMP_BTNCORNERS
Push-button corners
SBMP_MINBUTTON
Minimize button
SBMP_MINBUTTONDEP
Minimize button in depressed state
SBMP_MAXBUTTON
Maximize button
SBMP_MAXBUTTONDEP
Maximize button in depressed state
SBMP_RESTOREBUTTON
Restore button
SBMP_RESTOREBUTTONDEP

	Restore button in depressed state
SBMP_CHILDSYSMENU	System menu for child windows
SBMP_CHILDSYSMENUDEP	System menu for child windows in depressed state
SBMP_DRIVE	Drive
SBMP_FILE	File
SBMP_FOLDER	Folder
SBMP_TREEPLUS	Used by the file system to indicate that an entry in the directory can be expanded.
SBMP_TREEMINUS	Used by the file system to indicate that an entry in the directory can be collapsed.
SBMP_PROGRAM	Used by the file system to mark .EXE and .COM files.
SBMP_SIZEBOX	Used by some applications to display a sizebox in the bottom-right corner of a frame window.

WinGetSysBitmap Return Value - hbm

hbm ([HBITMAP](#)) - returns
System bit-map handle.

NULLHANDLE	Error occurred
Other	System bit-map handle.

WinGetSysBitmap - Parameters

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

ibm ([ULONG](#)) - input
System bit-map index value.

SBMP_SYSMENU	System menu
SBMP_SYSMENUDEP	System menu in depressed state
SBMP_SBUPARROW	Scroll bar up arrow
SBMP_SBUPARROWDEP	Scroll bar up arrow in depressed state
SBMP_SBUPARROWDIS	Scroll bar up arrow in disabled state
SBMP_SBDNARROW	Scroll bar down arrow
SBMP_SBDNARROWDEP	Scroll bar down arrow in depressed state
SBMP_SBDNARROWDIS	

SBMP_SBRGARROW	Scroll bar down arrow in disabled state
SBMP_SBRGARROWDEP	Scroll bar right arrow
SBMP_SBRGARROWDIS	Scroll bar right arrow in depressed state
SBMP_SBLFARROW	Scroll bar right arrow in disabled state
SBMP_SBLFARROWDEP	Scroll bar left arrow
SBMP_SBLFARROWDIS	Scroll bar left arrow in depressed state
SBMP_MENUCHECK	Scroll bar left arrow in disabled state
SBMP_MENUATTACHED	Menu check mark
SBMP_CHECKBOXES	Cascading menu mark
SBMP_COMBODOWN	Check box or radio button check marks
SBMP_BTNCORNERS	Combobox down arrow
SBMP_MINBUTTON	Push-button corners
SBMP_MINBUTTONDEP	Minimize button
SBMP_MAXBUTTON	Minimize button in depressed state
SBMP_MAXBUTTONDEP	Maximize button
SBMP_RESTOREBUTTON	Maximize button in depressed state
SBMP_RESTOREBUTTONDEP	Restore button
SBMP_CHILDSYSTEMMENU	Restore button in depressed state
SBMP_CHILDSYSTEMMENUDEP	System menu for child windows
SBMP_DRIVE	System menu for child windows in depressed state
SBMP_FILE	Drive
SBMP_FOLDER	File
SBMP_TREEPLUS	Folder
SBMP_TREEMINUS	Used by the file system to indicate that an entry in the directory can be expanded.
SBMP_PROGRAM	Used by the file system to indicate that an entry in the directory can be collapsed.
SBMP_SIZEBOX	Used by the file system to mark .EXE and .COM files.
	Used by some applications to display a sizebox in the bottom-right corner of a frame window.

hbm ([HBITMAP](#)) - returns
System bit-map handle.

NULLHANDLE	Error occurred
Other	System bit-map handle.

WinGetSysBitmap - Remarks

The bit map returned can be used for any of the normal bit-map operations. This function provides a new copy of the system bit map each time it is called. The application should release any bit maps it gets with this function by using `GpiDeleteBitmap`.

WinGetSysBitmap - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)

An invalid window handle was specified.

PMERR_PARAMETER_OUT_OF_RANGE (0x1003)

The value of a parameter was not within the defined valid range for that parameter.

PMERR_RESOURCE_NOT_FOUND (0x100A)

The specified resource identity could not be found.

WinGetSysBitmap - Related Functions

Related Functions

- [WinDrawBitmap](#)
 - [WinDrawBorder](#)
 - [WinDrawPointer](#)
 - [WinDrawText](#)
 - [WinFillRect](#)
 - [WinGetSysBitmap](#)
 - [WinInvertRect](#)
 - [WinQueryPresParam](#)
 - [WinRemovePresParam](#)
 - [WinScrollWindow](#)
 - [WinSetPresParam](#)
-

WinGetSysBitmap - Example Code

This example uses WinGetSysBitmap to retrieve the system defined handle for the menu check mark bit map during the window creation phase. The bit-map handle is then later used to draw the check mark in response to user selection of a menu item.

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#define INCL_WINPOINTERS      /* Window Pointer Functions */
#define INCL_WINMESSAGEGR     /* Window Message Functions */
#define INCL_WINMENUS         /* Window Menu Functions */
#include <os2.h>

HPS    hps;                /* presentation-space handle */
HBITMAP hbmCheck;          /* check mark bit-map handle */
HWND    hwndMenu;          /* menu handle */
USHORT  usItemId;          /* menu item id */
MPARAM  mp1;               /* Parameter 1 (menu item id) */
MPARAM  mp2;               /* Parameter 2 (menu handle) */
RECT    rcItem;            /* item border rectangle */

case WM_CREATE:
    /* obtain check mark bit-map handle */
    hbmCheck = WinGetSysBitmap(HWND_DESKTOP, SBMP_MENUCHECK);

case WM_MENUSELECT:
    usItemId = SHORT1FROMMP(mp1);
    hwndMenu = HWNDFROMMP(mp2);

    /* get rectangle of selected item */
```

```

WinSendMsg(hwndMenu,
            MM_QUERYITEMRECT,
            MPFROM2SHORT(usItemId, TRUE),
            MPFROMP(&rclItem));

/* draw the check mark in the lower left corner of item's
rectangle */
if (hbmCheck != NULL)
{
    WinDrawBitmap(hps,
                  hbmCheck,          /* check mark          */
                  NULL,             /* draw whole bit map */
                  (PPOINTL)&rclItem, /* bit-map destination */
                  0L,               /* ignored since color */
                  0L,               /* bit map            */
                  DBM_NORMAL);      /* draw normal size    */
}

```

WinGetSysBitmap - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinInflateRect

WinInflateRect - Syntax

This function expands a rectangle.

```

#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;    /* Anchor-block handle. */
PRECTL   prcl;   /* Rectangle to be expanded. */
LONG     cx;     /* Horizontal expansion. */
LONG     cy;     /* Vertical expansion. */
BOOL     rc;     /* Success indicator. */

rc = WinInflateRect(hab, prcl, cx, cy);

```

WinInflateRect Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinInflateRect Parameter - prcl

prcl ([PRECTL](#)) - in/out
Rectangle to be expanded.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

WinInflateRect Parameter - cx

cx ([LONG](#)) - input
Horizontal expansion.

WinInflateRect Parameter - cy

cy ([LONG](#)) - input
Vertical expansion.

WinInflateRect Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinInflateRect - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

prcl ([PRECTL](#)) - in/out
Rectangle to be expanded.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

cx ([LONG](#)) - input
Horizontal expansion.

cy ([LONG](#)) - input
Vertical expansion.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinInflateRect - Remarks

This function adjusts the size of the rectangle by applying the *cx* parameter horizontally at both vertical edges and the *cy* parameter vertically at both horizontal edges.

The *cx* parameter is subtracted from the left and added to the right of the rectangle, and the *cy* parameter is subtracted from the bottom and added to the top of the rectangle.

If the values of the *cx* and *cy* parameters are both positive, the rectangle is enlarged and surrounds the original rectangle. Conversely, if both these values are negative, the rectangle is reduced in size and is inset with respect to the original rectangle.

WinInflateRect - Related Functions

Related Functions

- [WinCopyRect](#)
- [WinEqualRect](#)
- [WinFillRect](#)
- [WinInflateRect](#)
- [WinIntersectRect](#)
- [WinIsRectEmpty](#)
- [WinOffsetRect](#)
- [WinPtInRect](#)
- [WinSetRect](#)
- [WinSetRectEmpty](#)
- [WinSubtractRect](#)
- [WinUnionRect](#)

WinInflateRect - Example Code

This example doubles the size of a rectangle if the mouse is double clicked (WM_BUTTON1DBLCLK) within the rectangle (WinPtInRect).

```

#define INCL_WINRECTANGLES      /* Window Rectangle Functions */
#include <os2.h>

BOOL fSuccess;                /* success indicator */
HAB hab;                      /* anchor-block handle */
RECTL prclRect1 = {0,0,100,100}; /* rectangle */
LONG lcx = 100;               /* Horizontal expansion */
LONG lcy = 100;               /* Vertical expansion */
POINTL ptl;                   /* current mouse position */
MPARAM mpParam1;              /* Parameter 1 (x,y) point value */

case WM_BUTTON1DBLCLK:
    ptl.x = (LONG) SHORT1FROMMP(mpParam1);
    ptl.y = (LONG) SHORT2FROMMP(mpParam1);

    if (WinPtInRect(hab, &prclRect1, &ptl))
        fSuccess = WinInflateRect(hab, &prclRect1, lcx, lcy);

```

WinInflateRect - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinInitialize

WinInitialize - Syntax

This function initializes the PM facilities for use by an application.

```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

ULONG fOptions; /* Initialization options. */
HAB hab;        /* Anchor-block handle. */

hab = WinInitialize(fOptions);

```

WinInitialize Parameter - fOptions

flOptions (**ULONG**) - input
Initialization options.

0

The initial state for newly created windows is that all messages for the window are available for processing by the application.

This is the only option available in PM.

WinInitialize Return Value - hab

hab (**HAB**) - returns
Anchor-block handle.

NULLHANDLE

An error occurred.

Other

Anchor-block handle.

WinInitialize - Parameters

flOptions (**ULONG**) - input
Initialization options.

0

The initial state for newly created windows is that all messages for the window are available for processing by the application.

This is the only option available in PM.

hab (**HAB**) - returns
Anchor-block handle.

NULLHANDLE

An error occurred.

Other

Anchor-block handle.

WinInitialize - Remarks

This must be the first PM call issued by any application thread using Presentation Manager facilities.

It returns *hab*, which is NULL if the initialization is not successful.

The operating system does not generally use the information supplied by the *hab* parameter to its calls; instead, it deduces it from the identity of the thread that is making the call. Thus an OS/2 application is not required to supply any particular value as the *hab* parameter. However, in order to be portable to other environments, an application must provide the *hab*, that is returned by the WinInitialize function of the thread, to any OS/2 function that requires it.

flOptions determines the initial state of message processing with respect to a created window.

WinInitialize - Related Functions

Related Functions

- [WinCancelShutdown](#)
 - [WinCreateMsgQueue](#)
 - [WinInitialize](#)
 - [WinTerminate](#)
-

WinInitialize - Example Code

This example uses WinInitialize to obtain an anchor block and initialize Presentation Manager.

```
#define INCL_WINMESSAGEGR      /* Window Message Functions */
#define INCL_WINWINDOWMGR     /* Window Manager Functions */
#include <os2.h>

HAB      hab;           /* anchor-block handle */
HMQ      hmq;           /* message queue handle */
QMSG     qmsg;          /* message */

hab = WinInitialize(0);      /* initialize PM */

hmq = WinCreateMsgQueue(hab, 0); /* create default size queue */

/*
 *
 *   initialize windows
 *
 */

/* get and dispatch messages from queue */
while (WinGetMsg(hab, &qmsg, 0, 0, 0))
    WinDispatchMsg(hab, &qmsg);
```

WinInitialize - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinInSendMessage

WinInSendMessage - Syntax

This function determines whether the current thread is processing a message sent by another thread.

```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB    hab; /* Anchor-block handle. */
BOOL   rc; /* Message-processing indicator. */

rc = WinInSendMessage(hab);
```

WinInSendMessage Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinInSendMessage Return Value - rc

rc (**BOOL**) - returns
Message-processing indicator.

TRUE	Current thread is processing a message sent by another thread
FALSE	Current thread is not processing a message, or an error occurred.

WinInSendMessage - Parameters

hab (**HAB**) - input
Anchor-block handle.

rc (**BOOL**) - returns
Message-processing indicator.

TRUE	Current thread is processing a message sent by another thread
FALSE	Current thread is not processing a message, or an error occurred.

WinInSendMessage - Remarks

If the message is from another thread this function determines whether or not the message was initiated by the active thread. The "active thread" is the thread associated with the current active window. (See also the [WinIsThreadActive](#) function.)

Typically this function is used by applications to determine how to proceed with errors when the window processing the message is not the active window. For example, if the active window uses the [WinSendMsg](#) function to send a request for information to another window, the other window cannot become active until it returns control from the [WinSendMsg](#) function. The only methods an inactive window has to inform the user of an error are to create a message box (see [WinMessageBox](#)), or to flash a window (see [WinFlashWindow](#)).

This function can be used to tell if a function is being called recursively.

WinInSendMessage - Related Functions

Related Functions

- [WinBroadcastMsg](#)
- [WinCreateMsgQueue](#)
- [WinDestroyMsgQueue](#)
- [WinDispatchMsg](#)
- [WinGetDlgMsg](#)
- [WinGetMsg](#)
- [WinInSendMessage](#)
- [WinPeekMsg](#)
- [WinPostMsg](#)
- [WinPostQueueMsg](#)
- [WinQueryMsgPos](#)
- [WinQueryMsgTime](#)
- [WinQueryQueueInfo](#)
- [WinQueryQueueStatus](#)
- [WinSendDlgItemMsg](#)
- [WinSendMsg](#)
- [WinSetClassMsgInterest](#)
- [WinSetMsgInterest](#)
- [WinSetMsgMode](#)
- [WinSetSynchroMode](#)
- [WinWaitMsg](#)

WinInSendMessage - Example Code

This example determines, during a WM_ERROR message, if the current thread is processing a message sent by another thread using WinInSendMessage; if so, a message box is generated with the error information to alert the active window that originally sent the message.

```
#define INCL_WINMESSAGEGR      /* Window Message Functions */
#define INCL_WINDIALOGS       /* Window Dialog Mgr Functions */
#include <os2.h>

HAB      hab;           /* anchor-block handle */
BOOL     fSuccess;      /* Success indicator */
MPARAM   mpParam1;      /* Parameter 1 */
USHORT   errorcode;     /* error code */
CHAR     szMsg[100];    /* message text */
HWND     hwnd;          /* handle of window with error msg */

case WM_ERROR:
    /* get error code */
    errorcode = SHORT1FROMMP(mpParam1);

    if (WinInSendMessage(hab))
    {
        /* parse and display error message */
        sprintf(szMsg, "Error code %d occurred", errorcode);
        WinMessageBox(HWND_DESKTOP,
```

```

        hwnd,                /* client-window handle */
        szMsg,               /* body of the message */
        "Error notification", /* title of the message */
        0,                  /* message box id */
        MB_NOICON | MB_OK); /* icon and button flags */
}

```

WinInSendMessage - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinInsertLboxItem

WinInsertLboxItem - Syntax

This macro inserts text into a list box at index, index may be a LIT_ constant. The macro returns the actual index where it was inserted.

```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndLbox; /* List box handle. */
LONG    index;    /* Index of the list box item. */
PSZ     psz;      /* Text to be inserted. */
LONG    lRetIndex; /* Actual index where it was inserted. */

lRetIndex = WinInsertLboxItem(hwndLbox, index,
                             psz);

```

WinInsertLboxItem Parameter - hwndLbox

hwndLbox ([HWND](#)) - input
List box handle.

WinInsertLboxItem Parameter - index

index (**LONG**) - input
Index of the list box item.

WinInsertLboxItem Parameter - psz

psz (**PSZ**) - input
Text to be inserted.

WinInsertLboxItem Return Value - IRetIndex

IRetIndex (**LONG**) - returns
Actual index where it was inserted.

WinInsertLboxItem - Parameters

hwndLbox (**HWND**) - input
List box handle.

index (**LONG**) - input
Index of the list box item.

psz (**PSZ**) - input
Text to be inserted.

IRetIndex (**LONG**) - returns
Actual index where it was inserted.

WinInsertLboxItem - Remarks

This macro is defined as:

```
#define WinInsertLboxItem(hwndLbox, index, psz) \
    ((LONG)WinSendMessage(hwndLbox, \
        LM_INSERTITEM, \
        MPFROMLONG(index), \
        MPFROMP(psz)))
```

This macro requires the existence of a message queue.

WinInsertLboxItem - Related Functions

Related Functions

- [WinSendMsg](#)
-

WinInsertLboxItem - Related Messages

Related Messages

- [LM_INSERTITEM](#)
-

WinInsertLboxItem - Example Code

This example calls WinInsertLboxItem to insert items in a list box as part of initializing a dialog (WM_INITDLG message).

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#define INCL_WINLISTBOXES     /* Window List Box definitions */
#include <os2.h>

LONG      lIndex;              /* inserted item index */
HWND      hwndLbox;           /* list box window handle */
MPARAM    mpParam1;           /* Parameter 1 (window handle) */
/* Array of list box item names */
PSZ       pszItems[3] = {"Item1", "Item2", "Item3"};

case WM_INITDLG:
    .
    .
    .
    /*****
    /* Initialize List Box Control */
    *****/

    /* get handle of list box */
    hwndLbox = HWNDFROMMP(mpParam1);

    /* insert 3 items into list box */
    lIndex = WinInsertLboxItem(hwndLbox, LIT_END, pszItems[0]);
    lIndex = WinInsertLboxItem(hwndLbox, LIT_END, pszItems[1]);
    lIndex = WinInsertLboxItem(hwndLbox, LIT_END, pszItems[2]);
    .
    .
```

WinInsertLboxItem - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinIntersectRect

WinIntersectRect - Syntax

This function calculates the intersection of the two source rectangles and returns the result in the destination rectangle.

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
PRECTL   pcrIDst;      /* Intersection rectangle. */
PRECTL   pcrISrc1;     /* First rectangle. */
PRECTL   pcrISrc2;     /* Second rectangle. */
BOOL      rc;          /* Success indicator. */

rc = WinIntersectRect(hab, pcrIDst, pcrISrc1,
                      pcrISrc2);
```

WinIntersectRect Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinIntersectRect Parameter - pcrIDst

pcrIDst ([PRECTL](#)) - output
Intersection rectangle.

Is the intersection of *pcrISrc1* and *pcrISrc2*.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

WinIntersectRect Parameter - pcrISrc1

pcrISrc1 ([PRECTL](#)) - input
First rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

WinIntersectRect Parameter - pcrISrc2

pcrISrc2 ([PRECTL](#)) - input
Second rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

WinIntersectRect Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

- | | |
|-------|---|
| TRUE | Source rectangles intersect |
| FALSE | Source rectangles do not intersect, or an error occurred. |

WinIntersectRect - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

pcrIDst ([PRECTL](#)) - output
Intersection rectangle.

Is the intersection of *pcrISrc1* and *pcrISrc2*.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

pcrISrc1 ([PRECTL](#)) - input
First rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

pcrISrc2 ([PRECTL](#)) - input
Second rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

rc ([BOOL](#)) - returns
Success indicator.

TRUE
Source rectangles intersect

FALSE
Source rectangles do not intersect, or an error occurred.

WinIntersectRect - Remarks

If there is no intersection, an empty rectangle is returned in *prclDst*.

WinIntersectRect - Related Functions

Related Functions

- [WinCopyRect](#)
 - [WinEqualRect](#)
 - [WinFillRect](#)
 - [WinInflateRect](#)
 - [WinIntersectRect](#)
 - [WinIsRectEmpty](#)
 - [WinOffsetRect](#)
 - [WinPtInRect](#)
 - [WinSetRect](#)
 - [WinSetRectEmpty](#)
 - [WinSubtractRect](#)
 - [WinUnionRect](#)
-

WinIntersectRect - Example Code

This example determines the intersection of two rectangles and places the result in a third rectangle structure.

```
#define INCL_WINRECTANGLES      /* Window Rectangle Functions */
#include <os2.h>

BOOL    fSuccess;              /* success indicator          */
HAB     hab;                   /* anchor-block handle       */
PRECTL  prclRect1 = {0,0,100,100}; /* rectangle 1               */
PRECTL  prclRect2 = {0,0,200,200}; /* rectangle 2               */
PRECTL  prclDest;              /* destination rectangle      */

fSuccess = WinIntersectRect(hab, &prclDest, &prclRect1,
                           &prclRect2);
```

WinIntersectRect - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinInvalidateRect

WinInvalidateRect - Syntax

This function adds a rectangle to a window's update region.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwnd;           /* Handle of window whose update region is to be changed. */
PRECTL    pwrcl;          /* Update rectangle. */
BOOL      fIncludeChildren; /* Invalidation-scope indicator. */
BOOL      rc;             /* Success indicator. */

rc = WinInvalidateRect(hwnd, pwrcl, fIncludeChildren);
```

WinInvalidateRect Parameter - hwnd

hwnd (**HWND**) - input

Handle of window whose update region is to be changed.

HWND_DESKTOP

This function applies to the whole screen (or desktop).

Other

Handle of window whose update region is to be changed.

WinInvalidateRect Parameter - pwrcl

pwrcl (**PRECTL**) - input

Update rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

NULL	The whole window is to be added into the window's update region.
Other	Rectangle to be added to the window's update region.

WinInvalidateRect Parameter - fIncludeChildren

fIncludeChildren ([BOOL](#)) - input
Invalidation-scope indicator.

TRUE	Include the descendants of <i>hwnd</i> in the invalid rectangle.
FALSE	Include the descendants of <i>hwnd</i> in the invalid rectangle, but only if the parent does not have a WS_CLIPCHILDREN style.

WinInvalidateRect Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinInvalidateRect - Parameters

hwnd ([HWND](#)) - input
Handle of window whose update region is to be changed.

HWND_DESKTOP	This function applies to the whole screen (or desktop).
Other	Handle of window whose update region is to be changed.

pwrcl ([PRECTL](#)) - input
Update rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

NULL	The whole window is to be added into the window's update region.
Other	Rectangle to be added to the window's update region.

flIncludeChildren ([BOOL](#)) - input
Invalidation-scope indicator.

TRUE

Include the descendants of *hwnd* in the invalid rectangle.

FALSE

Include the descendants of *hwnd* in the invalid rectangle, but only if the parent does not have a WS_CLIPCHILDREN style.

rc ([BOOL](#)) - returns
Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinInvalidateRect - Remarks

The update region is a subregion of a window that is deemed "invalid" or incorrect in visual terms and in need of redrawing.

If the window has a CS_SYNCPAINT style, it is redrawn during the processing of this function and the update region should be NULL on return from this function.

If the window has a WS_CLIPCHILDREN style with part of its update region overlapping child windows with a CS_SYNCPAINT style, those children are updated before this function returns.

This function should not be called in response to a [WM_PAINT](#) request for windows of style CS_SYNCPAINT. CS_SYNCPAINT means that windows are updated synchronously when invalidated, which generates a [WM_PAINT](#) message. Thus, invalidating the window in response to a [WM_PAINT](#) message would cause another invalidate, and another [WM_PAINT](#), and so on.

WinInvalidateRect - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_INVALID_FLAG (0x1019)
An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

WinInvalidateRect - Related Functions

Related Functions

- [WinBeginPaint](#)
- [WinEnableWindowUpdate](#)
- [WinEndPaint](#)
- [WinExcludeUpdateRegion](#)
- [WinGetClipPS](#)
- [WinGetPS](#)
- [WinGetScreenPS](#)
- [WinInvalidateRect](#)
- [WinInvalidateRegion](#)
- [WinIsWindowShowing](#)

- [WinIsWindowVisible](#)
- [WinLockVisRegions](#)
- [WinOpenWindowDC](#)
- [WinQueryUpdateRect](#)
- [WinQueryUpdateRegion](#)
- [WinRealizePalette](#)
- [WinReleasePS](#)
- [WinShowWindow](#)
- [WinUpdateWindow](#)
- [WinValidateRect](#)
- [WinValidateRegion](#)

WinInvalidateRect - Related Messages

Related Messages

- [WM_PAINT](#)

WinInvalidateRect - Example Code

This example gets the dimensions of the window and calls WinInvalidateRect to invalidate the window. The application will be sent a WM_PAINT message with the entire window as the update rectangle.

```
#define INCL_WINWINDOWMGR          /* Window Manager Functions */
#include <os2.h>

HWND    hwnd;                    /* parent window */
RECT    rcl;                     /* update region */

WinQueryWindowRect(hwnd, &rcl);

WinInvalidateRect(hwnd,          /* window to invalidate */
                  &rcl,          /* invalid rectangle */
                  FALSE);        /* do not include children */
```

WinInvalidateRect - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinInvalidateRegion

WinInvalidateRegion - Syntax

This function adds a region to a window's update region.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd;          /* Handle of window whose update region is to be changed. */
HRGN    hrgn;          /* Handle of the region to be added to the update region of the window. */
BOOL    fIncludeChildren; /* Invalidation-scope indicator. */
BOOL    rc;            /* Success indicator. */

rc = WinInvalidateRegion(hwnd, hrgn, fIncludeChildren);
```

WinInvalidateRegion Parameter - hwnd

hwnd (**HWND**) - input

Handle of window whose update region is to be changed.

HWND_DESKTOP

This function applies to the whole screen (or desktop).

Other

Handle of window whose update region is to be changed.

WinInvalidateRegion Parameter - hrgn

hrgn (**HRGN**) - input

Handle of the region to be added to the update region of the window.

NULLHANDLE

The whole window is to be added into the window's update region.

Other

Handle of the region to be added to the window's update region.

WinInvalidateRegion Parameter - fIncludeChildren

fIncludeChildren (**BOOL**) - input

Invalidation-scope indicator.

TRUE

FALSE	Include the descendants of <i>hwnd</i> in the invalid rectangle.
	Include the descendants of <i>hwnd</i> in the invalid rectangle, but only if the parent does not have a WS_CLIPCHILDREN style.

WinInvalidateRegion Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinInvalidateRegion - Parameters

hwnd (HWND) - input
Handle of window whose update region is to be changed.

HWND_DESKTOP	This function applies to the whole screen (or desktop).
Other	Handle of window whose update region is to be changed.

hrgn (HRGN) - input
Handle of the region to be added to the update region of the window.

NULLHANDLE	The whole window is to be added into the window's update region.
Other	Handle of the region to be added to the window's update region.

IncludeChildren (BOOL) - input
Invalidation-scope indicator.

TRUE	Include the descendants of <i>hwnd</i> in the invalid rectangle.
FALSE	Include the descendants of <i>hwnd</i> in the invalid rectangle, but only if the parent does not have a WS_CLIPCHILDREN style.

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinInvalidateRegion - Remarks

The update region is a subregion of a window that is deemed "invalid" or incorrect in visual terms and is in need of redrawing.

If the window has a `CS_SYNCPAINT` style, it is redrawn during the processing of this function and the update region should be `NULL` on return from this function.

If the window has a `WS_CLIPCHILDREN` style with part of its update region overlapping child windows with a `CS_SYNCPAINT` style, those children are updated before this function returns.

This function should not be called in response to a [WM_PAINT](#) request for windows of style `CS_SYNCPAINT`. `CS_SYNCPAINT` means that windows are updated synchronously when invalidated, which generates a [WM_PAINT](#) message. Thus, invalidating the window in response to a [WM_PAINT](#) message would cause another invalidate, another [WM_PAINT](#), and so on.

WinInvalidateRegion - Errors

Possible returns from [WinGetLastError](#)

`PMERR_INVALID_HWND` (0x1001)
An invalid window handle was specified.

`PMERR_HRGN_BUSY` (0x2034)
An internal region busy error was detected. The region was locked by one thread during an attempt to access it from another thread.

`PMERR_INVALID_FLAG` (0x1019)
An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

WinInvalidateRegion - Related Functions

Related Functions

- [WinBeginPaint](#)
- [WinEnableWindowUpdate](#)
- [WinEndPaint](#)
- [WinExcludeUpdateRegion](#)
- [WinGetClipPS](#)
- [WinGetPS](#)
- [WinGetScreenPS](#)
- [WinInvalidateRect](#)
- [WinInvalidateRegion](#)
- [WinIsWindowShowing](#)
- [WinIsWindowVisible](#)
- [WinLockVisRegions](#)
- [WinOpenWindowDC](#)
- [WinQueryUpdateRect](#)
- [WinQueryUpdateRegion](#)
- [WinRealizePalette](#)
- [WinReleasePS](#)
- [WinShowWindow](#)
- [WinUpdateWindow](#)
- [WinValidateRect](#)
- [WinValidateRegion](#)

WinInvalidateRegion - Related Messages

Related Messages

- [WM_PAINT](#)

WinInvalidateRegion - Example Code

This example invalidates the entire window by adding the whole window to the window's update region using WinInvalidateRegion. This single call accomplishes the same as paired calls to WinQueryWindowRect and WinInvalidateRect. If less than the entire window is desired, the NULL value in the second parameter can be replaced with a region handle that corresponds to a subregion of the window.

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#include <os2.h>

HWND      hwnd;                /* window handle */

WinInvalidateRegion(hwnd, NULLHANDLE, 0);
```

WinInvalidateRegion - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Errors](#)
- [Remarks](#)
- [Example Code](#)
- [Related Functions](#)
- [Related Messages](#)
- [Glossary](#)

WinInvertRect

WinInvertRect - Syntax

This function inverts a rectangular area.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HPS      hps;                /* Presentation-space handle. */
PRECTL   prclRect;          /* Rectangle to be inverted. */
BOOL      rc;                /* Success indicator. */

rc = WinInvertRect(hps, prclRect);
```

WinInvertRect Parameter - hps

hps ([HPS](#)) - input
Presentation-space handle.

The presentation space contains the rectangle to be inverted.

WinInvertRect Parameter - prclRect

prclRect ([PRECTL](#)) - input
Rectangle to be inverted.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

WinInvertRect Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinInvertRect - Parameters

hps ([HPS](#)) - input
Presentation-space handle.

The presentation space contains the rectangle to be inverted.

prclRect ([PRECTL](#)) - input
Rectangle to be inverted.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinInvertRect - Remarks

Inversion is a logical-NOT operation and has the effect of flipping the bits of each pel.

WinInvertRect - Related Functions

Related Functions

- [WinDrawBitmap](#)
 - [WinDrawBorder](#)
 - [WinDrawPointer](#)
 - [WinDrawText](#)
 - [WinFillRect](#)
 - [WinGetSysBitmap](#)
 - [WinInvertRect](#)
 - [WinQueryPresParam](#)
 - [WinRemovePresParam](#)
 - [WinScrollWindow](#)
 - [WinSetPresParam](#)
-

WinInvertRect - Example Code

This example inverts a rectangle if the mouse button is released (WM_BUTTON1UP) within the rectangle (WinPtInRect); the presentation space handle is obtained via WinBeginPaint.

```
#define INCL_WINWINDOWMGR          /* Window Manager Functions */
#include <os2.h>

BOOL fSuccess;                    /* success indicator */
HAB hab;                          /* anchor-block handle */
RECTL prclRect1 = {0,0,100,100}; /* rectangle */
HWND hwnd;                       /* client window handle */
HPS hps;                         /* presentation-space handle */
POINTL ptl;                      /* current mouse position */
MPARAM mpParam1;                 /* Parameter 1 (x,y) point value */

case WM_BUTTON1UP:
    ptl.x = (LONG) SHORT1FROMMP(mpParam1);
    ptl.y = (LONG) SHORT2FROMMP(mpParam1);

    if (WinPtInRect(hab, &prclRect1, &ptl))
    {
        hps = WinBeginPaint(hwnd, NULLHANDLE, &prclRect1);
        fSuccess = WinInvertRect(hps, &prclRect1);
        WinEndPaint(hps);
    }
```

WinInvertRect - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinIsChild

WinIsChild - Syntax

This function indicates if a window is a descendant of another window.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd;           /* Child-window handle. */
HWND    hwndParent;     /* Parent-window handle. */
BOOL    fRelated;       /* Related indicator. */

fRelated = WinIsChild(hwnd, hwndParent);
```

WinIsChild Parameter - hwnd

hwnd ([HWND](#)) - input
Child-window handle.

WinIsChild Parameter - hwndParent

hwndParent ([HWND](#)) - input
Parent-window handle.

WinIsChild Return Value - fRelated

fRelated ([BOOL](#)) - returns
Related indicator.

TRUE

Child window is a descendant of the parent window, or is equal to it

FALSE

Child window is not a descendant of the parent, or is an Object Window (even if *hwndParent* is specified as the desktop or `HWND_DESKTOP`), or an error occurred.

WinIsChild - Parameters

hwnd ([HWND](#)) - input
Child-window handle.

hwndParent ([HWND](#)) - input
Parent-window handle.

fRelated ([BOOL](#)) - returns
Related indicator.

TRUE

Child window is a descendant of the parent window, or is equal to it

FALSE

Child window is not a descendant of the parent, or is an Object Window (even if *hwndParent* is specified as the desktop or `HWND_DESKTOP`), or an error occurred.

WinIsChild - Errors

Possible returns from [WinGetLastError](#)

`PMERR_INVALID_HWND (0x1001)`
An invalid window handle was specified.

WinIsChild - Related Functions

Related Functions

- [WinBeginEnumWindows](#)
 - [WinEndEnumWindows](#)
 - [WinEnumDlgItem](#)
 - [WinGetNextWindow](#)
 - [WinIsChild](#)
 - [WinMultWindowFromIDs](#)
 - [WinQueryWindow](#)
 - [WinSetOwner](#)
 - [WinSetParent](#)
-

WinIsChild - Example Code

This example uses WinIsChild to determine if one window is a descendant of another window.

```
#define INCL_WINWINDOWMGR    /* Window Manager Functions */
#include <os2.h>

HWND    hwndChild;          /* Child window to check    */
HWND    hwndParent;         /* Parent window to check   */

if (WinIsChild(hwndChild, hwndParent))
{
    /* hwndChild is a descendant of hwndParent */
}
else
{
    /* hwndChild is not a descendant of hwndParent */
}
```

WinIsChild - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinIsControlEnabled

WinIsControlEnabled - Syntax

This function returns the state (enabled/disabled) of the specified item in the dialog template within a dialog box.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndDlg; /* Dialog window handle. */
USHORT  usId;    /* Identity of the specified item. */
BOOL    rc;      /* Enabled-state indicator. */

rc = WinIsControlEnabled(hwndDlg, usId);
```

WinIsControlEnabled Parameter - hwndDlg

hwndDlg (**HWND**) - input
Dialog window handle.

WinIsControlEnabled Parameter - usId

usId (**USHORT**) - input
Identity of the specified item.

WinIsControlEnabled Return Value - rc

rc (**BOOL**) - returns
Enabled-state indicator.

- | | |
|-------|-----------------------------|
| TRUE | Specified item is enabled. |
| FALSE | Specified item is disabled. |
-

WinIsControlEnabled - Parameters

hwndDlg (**HWND**) - input
Dialog window handle.

usId (**USHORT**) - input
Identity of the specified item.

rc (**BOOL**) - returns
Enabled-state indicator.

- | | |
|-------|-----------------------------|
| TRUE | Specified item is enabled. |
| FALSE | Specified item is disabled. |
-

WinIsControlEnabled - Remarks

This macro expands to:

```
#define WinIsControlEnabled(hwndDlg, usId)  
((BOOL)WinIsWindowEnabled(WinWindowFromID(hwndDlg, usId)))
```

This function requires the existence of a message queue.

WinIsControlEnabled - Related Functions

Related Functions

- [WinIsWindowEnabled](#)
- [WinWindowFromID](#)

WinIsControlEnabled - Example Code

This example uses WinIsControlEnabled to determine if a selected control is valid; if it is not, an error message box is displayed.

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions          */
#define INCL_WINDIALOGS       /* Window Dialog Mgr Functions      */
#include <os2.h>

HWND    hwndDlg;              /* Dialog window                    */
MPARAM  mp1;                  /* Parameter 1                      */
USHORT  usId;                 /* Dialog control id                */

case WM_CONTROL:
    usId = SHORT1FROMMP(mp1);
    if (!WinIsControlEnabled(hwndDlg, usId))
    {
        WinMessageBox(HWND_DESKTOP,
                       hwndDlg,
                       "Control is not valid",
                       "Error notification",
                       0,
                       MB_NOICON |
                       MB_OK);
    }
```

WinIsControlEnabled - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Example Code](#)
 - [Related Functions](#)
 - [Glossary](#)

WinIsMenuItemChecked

WinIsMenuItemChecked - Syntax

This function returns the state (checked/not checked) of the identified menu item.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND     hwndMenu; /* Menu window handle. */
USHORT   usId;     /* Identity of the menu item. */
BOOL     rc;       /* Checked-state indicator. */

rc = WinIsMenuItemChecked(hwndMenu, usId);
```

WinIsMenuItemChecked Parameter - hwndMenu

hwndMenu (**HWND**) - input
Menu window handle.

WinIsMenuItemChecked Parameter - usId

usId (**USHORT**) - input
Identity of the menu item.

WinIsMenuItemChecked Return Value - rc

rc (**BOOL**) - returns
Checked-state indicator.

TRUE	Menu item is checked.
FALSE	Menu item is not checked.

WinIsMenuItemChecked - Parameters

hwndMenu (**HWND**) - input
Menu window handle.

usId ([USHORT](#)) - input
Identity of the menu item.

rc ([BOOL](#)) - returns
Checked-state indicator.

TRUE	Menu item is checked.
FALSE	Menu item is not checked.

WinIsMenuItemChecked - Remarks

This macro expands to:

```
#define WinIsMenuItemChecked(hwndMenu, usId)
( (BOOL)WinSendMsg(hwndMenu,
    MM_QUERYITEMATTR,
    MPFROM2SHORT(usId, TRUE),
    MPFROM2SHORT(MIA_CHECKED)) )
```

This function requires the existence of a message queue.

WinIsMenuItemChecked - Related Functions

Related Functions

- [WinSendMsg](#)

WinIsMenuItemChecked - Related Messages

Related Messages

- [MM_QUERYITEMATTR](#)

WinIsMenuItemChecked - Example Code

This example uses WinIsMenuItemChecked to query the check attribute of a selected menu item before setting the check state of that menu item.

```
#define INCL_WINWINDOWMGR    /* Window Manager Functions */
#include <os2.h>

USHORT  usItemId;           /* Menu item ID */
HWND    hwndMenu;          /* Menu handle */
BOOL     usChkstate;        /* New checked state */
```

```

BOOL      fSuccess;          /* Success indicator */
MPARAM    mp1;               /* Parameter 1 (menu item ID) */
MPARAM    mp2;               /* Parameter 2 (menu handle) */

case WM_MENUSELECT:
    usItemId = SHORT1FROMMP(mp1);
    hwndMenu = HWNDFROMMP(mp2);

    /* Query the current check state */
    usChkstate = WinIsMenuItemChecked(hwndMenu, usItemId);

    /* Set the menu item check state */
    fSuccess = WinCheckMenuItem(hwndMenu, usItemId, usChkstate);

```

WinIsMenuItemChecked - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinIsMenuItemEnabled

WinIsMenuItemEnabled - Syntax

This function returns the state (enabled/disabled) of the specified menu item.

```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndMenu; /* Menu window handle. */
USHORT    usId;     /* Identity of the menu item. */
BOOL      rc;       /* Enabled-state indicator. */

rc = WinIsMenuItemEnabled(hwndMenu, usId);

```

WinIsMenuItemEnabled Parameter - hwndMenu

hwndMenu (**HWND**) - input
Menu window handle.

WinIsMenuItemEnabled Parameter - usId

usId (**USHORT**) - input
Identity of the menu item.

WinIsMenuItemEnabled Return Value - rc

rc (**BOOL**) - returns
Enabled-state indicator.

TRUE	Menu item is enabled.
FALSE	Menu item is disabled.

WinIsMenuItemEnabled - Parameters

hwndMenu (**HWND**) - input
Menu window handle.

usId (**USHORT**) - input
Identity of the menu item.

rc (**BOOL**) - returns
Enabled-state indicator.

TRUE	Menu item is enabled.
FALSE	Menu item is disabled.

WinIsMenuItemEnabled - Remarks

This macro expands to:

```
#define WinIsMenuItemEnabled(hwndMenu, usId)
( !(BOOL)WinSendMsg(hwndMenu,
    MM_QUERYITEMATTR,
    MPFROM2SHORT(usId, TRUE),
    MPFROMSHORT(MIA_DISABLED)) )
```

This function requires the existence of a message queue.

This function returns TRUE if called with an invalid window handle or menu item. The window handle and menu item should be separately validated before being passed as parameters to this function.

WinIsMenuItemEnabled - Related Functions

Related Functions

- [WinSendMsg](#)

WinIsMenuItemEnabled - Related Messages

Related Messages

- [MM_QUERYITEMATTR](#)

WinIsMenuItemEnabled - Example Code

This example uses WinIsMenuItemEnabled to determine if a selected menu item is available for use. If the item is not valid (WinIsMenuItemValid) or not enabled, a beep is emitted.

```
#define INCL_WINMESSAGEGR      /* Window Message Functions */
#define INCL_WINMENUS          /* Window Menu Functions */
#define INCL_DOSPROCESS        /* OS/2 Process Functions */
#include <os2.h>

MPARAM mp1;                    /* Parameter 1 (rectl structure) */
MPARAM mp2;                    /* Parameter 2 (frame boolean) */
USHORT usItemId;               /* Menu item ID */
HWND hwndMenu;                /* Menu handle */

case WM_MENUSELECT:
    usItemId = SHORT1FROMMP(mp1);
    hwndMenu = HWNDFROMMP(mp2);

    /* If the menu item is not valid or enabled, emit a beep */
    if (!WinIsMenuItemValid(hwndMenu, usItemId) ||
        !WinIsMenuItemEnabled(hwndMenu, usItemId))
        DosBeep(800,100L);
```

WinIsMenuItemEnabled - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinIsMenuItemValid

WinIsMenuItemValid - Syntax

This function indicates if the specified menu item is a valid choice.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND     hwndMenu; /* Menu window handle. */
USHORT   usId;      /* Identity of the menu item. */
BOOL     rc;        /* Validity indicator. */

rc = WinIsMenuItemValid(hwndMenu, usId);
```

WinIsMenuItemValid Parameter - hwndMenu

hwndMenu ([HWND](#)) - input
Menu window handle.

WinIsMenuItemValid Parameter - usId

usId ([USHORT](#)) - input
Identity of the menu item.

WinIsMenuItemValid Return Value - rc

rc ([BOOL](#)) - returns
Validity indicator.

TRUE	Menu item is valid.
FALSE	Menu item is invalid.

WinIsMenuItemValid - Parameters

hwndMenu ([HWND](#)) - input
Menu window handle.

usId ([USHORT](#)) - input
Identity of the menu item.

rc ([BOOL](#)) - returns
Validity indicator.

TRUE	Menu item is valid.
FALSE	Menu item is invalid.

WinIsMenuItemValid - Remarks

This macro expands to:

```
#define WinIsMenuItemValid(hwndMenu, usId)
( (BOOL)WinSendMsg(hwndMenu,
                    MM_ISITEMVALID,
                    MPFROM2SHORT(usId, TRUE),
                    MPFROMSHORT(FALSE)) )
```

This function requires the existence of a message queue.

WinIsMenuItemValid - Related Functions

Related Functions

- [WinSendMsg](#)

WinIsMenuItemValid - Related Messages

Related Messages

- [MM_ISITEMVALID](#)

WinIsMenuItemValid - Example Code

This example uses WinIsMenuItemValid to determine if a selected menu item is available for use. If the item is disabled (WinIsMenuItemEnabled) or invalid, a beep is emitted.

```
#define INCL_WINMESSAGEGR  /* Window Message Functions */
#define INCL_WINMENUS      /* Window Menu Functions      */
#define INCL_DOSPROCESS    /* OS/2 Process Functions */
#include <os2.h>

MPARAM mp1;          /* Parameter 1 (rectl structure) */
MPARAM mp2;          /* Parameter 2 (frame boolean)   */
USHORT usItemId;     /* Menu item ID                  */
HWND   hwndMenu;     /* Menu handle                    */

case WM_MENUSELECT:
    usItemId = SHORT1FROMMP(mp1);
    hwndMenu = HWNDFROMMP(mp2);

    /* If the menu item is invalid or disabled, emit a beep */
    if (!WinIsMenuItemValid(hwndMenu, usItemId) ||
        !WinIsMenuItemEnabled(hwndMenu, usItemId))
        DosBeep(800,100L);
```

WinIsMenuItemValid - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinIsPhysInputEnabled

WinIsPhysInputEnabled - Syntax

This function returns the state (enabled/disabled) of physical input.

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND   hwndDeskTop; /* Desktop-window handle. */
```



```
BOOL rc; /* Enabled-state indicator. */  
rc = WinIsPhysInputEnabled(hwndDesktop);
```

WinIsPhysInputEnabled Parameter - hwndDesktop

hwndDesktop (HWND) - input
Desktop-window handle.

HWND_DESKTOP
The desktop-window handle

WinIsPhysInputEnabled Return Value - rc

rc (BOOL) - returns
Enabled-state indicator.

TRUE
Input is enabled.
FALSE
Input is disabled.

WinIsPhysInputEnabled - Parameters

hwndDesktop (HWND) - input
Desktop-window handle.

HWND_DESKTOP
The desktop-window handle

rc (BOOL) - returns
Enabled-state indicator.

TRUE
Input is enabled.
FALSE
Input is disabled.

WinIsPhysInputEnabled - Related Functions

Related Functions

- [WinEnablePhysInput](#)

WinIsPhysInputEnabled - Example Code

This example uses WinIsPhysInputEnabled to determine if physical input is enabled; if it is not, then WinEnablePhysInput is called to enable it.

```
#define INCL_WININPUT    /* Window Input Functions */
#include <os2.h>

if (!WinIsPhysInputEnabled(HWND_DESKTOP))
    /* Enable queuing of physical input */
    WinEnablePhysInput(HWND_DESKTOP, TRUE);
```

WinIsPhysInputEnabled - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinIsRectEmpty

WinIsRectEmpty - Syntax

This function checks whether a rectangle is empty.

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;      /* Anchor-block handle. */
PRECTL   prclprc;  /* Rectangle to be checked. */
BOOL     rc;        /* Empty indicator. */

rc = WinIsRectEmpty(hab, prclprc);
```

WinIsRectEmpty Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinIsRectEmpty Parameter - prclprc

prclprc ([PRECTL](#)) - input
Rectangle to be checked.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

WinIsRectEmpty Return Value - rc

rc ([BOOL](#)) - returns
Empty indicator.

- | | |
|-------|-------------------------|
| TRUE | Rectangle is empty |
| FALSE | Rectangle is not empty. |

WinIsRectEmpty - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

prclprc ([PRECTL](#)) - input
Rectangle to be checked.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) can also be used, if supported by the language.

rc ([BOOL](#)) - returns
Empty indicator.

- | | |
|-------|-------------------------|
| TRUE | Rectangle is empty |
| FALSE | Rectangle is not empty. |

WinIsRectEmpty - Remarks

A rectangle has area if its left edge coordinate is less than its right edge coordinate, and its bottom edge coordinate is less than its top edge

coordinate. An empty rectangle is one with no area.

WinIsRectEmpty - Related Functions

Related Functions

- [WinCopyRect](#)
 - [WinEqualRect](#)
 - [WinFillRect](#)
 - [WinInflateRect](#)
 - [WinIntersectRect](#)
 - [WinIsRectEmpty](#)
 - [WinOffsetRect](#)
 - [WinPtInRect](#)
 - [WinSetRect](#)
 - [WinSetRectEmpty](#)
 - [WinSubtractRect](#)
 - [WinUnionRect](#)
-

WinIsRectEmpty - Example Code

This example checks if a rectangle is empty (i.e. it has no area).

```
#define INCL_WINRECTANGLES      /* Window Rectangle Functions */
#include <os2.h>

BOOL fEmpty;                  /* empty indicator */
HAB hab;                      /* anchor-block handle */
RECTL prclRect1 = {0,0,100,100}; /* rectangle */

fEmpty = WinIsRectEmpty(hab, &prclRect1);
```

WinIsRectEmpty - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinIsSOMDDReady

WinIsSOMDDReady - Syntax

This function returns the state of the DSOM daemon (SOMDD), started by the Workplace Shell process using [WinRestartSOMDD](#).

```
#define INCL_WPCCLASS
#include <os2.h>

BOOL    fReady; /* SOMDD-state indicator. */

fReady = WinIsSOMDDReady();
```

WinIsSOMDDReady Return Value - fReady

fReady (BOOL) - returns
SOMDD-state indicator.

TRUE	SOMDD has been started by the Workplace Shell process.
FALSE	SOMDD has not been started by the Workplace Shell process.

WinIsSOMDDReady - Parameters

fReady (BOOL) - returns
SOMDD-state indicator.

TRUE	SOMDD has been started by the Workplace Shell process.
FALSE	SOMDD has not been started by the Workplace Shell process.

WinIsSOMDDReady - Remarks

This function returns the state of the DSOM daemon started only by the Workplace Shell process using a call to [WinRestartSOMDD](#). This does not include the status of the DSOM daemon if started by any other process.

Note: This function requires that the Presentation Manager Shell is up and running.

WinIsSOMDDReady - Related Functions

Related Functions

- [WinIsWPDServerReady](#)
- [WinRestartSOMDD](#)
- [WinRestartWPDServer](#)

WinIsSOMDDReady - Example Code

This example starts the DSOM daemon and, a short time later, checks to see if it indeed has started successfully.

```
#define INCL_WPCCLASS
#include <os2.h>

enum {OFF, ON};

WinRestartSOMDD(ON);
.
.
.
if ( WinIsSOMDDReady() )
    somPrintf ("SOMDD is running\n")
else
    somPrintf ("SOMDD failed to start, possibly started already"
               "by another process\n");
```

WinIsSOMDDReady - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinIsThreadActive

WinIsThreadActive - Syntax

This function determines whether the active window belongs to the calling execution thread.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
```

```
HAB    hab; /* Anchor-block handle of calling thread. */
BOOL   rc; /* Active-window indicator. */

rc = WinIsThreadActive(hab);
```

WinIsThreadActive Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle of calling thread.

WinIsThreadActive Return Value - rc

rc ([BOOL](#)) - returns
Active-window indicator.

TRUE	Active window belongs to the calling thread
FALSE	Active window does not belong to the calling thread.

WinIsThreadActive - Parameters

hab ([HAB](#)) - input
Anchor-block handle of calling thread.

rc ([BOOL](#)) - returns
Active-window indicator.

TRUE	Active window belongs to the calling thread
FALSE	Active window does not belong to the calling thread.

WinIsThreadActive - Related Functions

Related Functions

- [WinEnableWindow](#)
- [WinIsThreadActive](#)
- [WinIsWindow](#)
- [WinIsWindowEnabled](#)
- [WinQueryDesktopWindow](#)
- [WinQueryObjectWindow](#)
- [WinQueryWindowDC](#)

- [WinQueryWindowProcess](#)
- [WinQueryWindowRect](#)
- [WinWindowFromDC](#)
- [WinWindowFromID](#)
- [WinWindowFromPoint](#)

WinIsThreadActive - Example Code

This example uses WinIsThreadActive to verify that the active window belongs to the current thread before querying and enabling the system menu window via WinIsWindowEnabled and WinEnableWindow.

```
#define INCL_WINWINDOWMGR    /* Window Manager Functions */
#define INCL_WINFRAMEMGR    /* Window Frame Functions */
#include <os2.h>

HAB    hab;                /* Anchor-block handle */
HWND    hwndSysmenu;       /* System menu window */
HWND    hwnd;              /* Parent window */
BOOL    fSuccess;          /* Success indicator */

/* If the active window belongs to the current thread, */
/* query the enabled status of the system menu */
if (WinIsThreadActive(hab))
{
    /* Obtain the handle for the system menu */
    hwndSysmenu = WinWindowFromID(hwnd, FID_SYSMENU);

    /* If the system menu is disabled, enable it */
    if (!WinIsWindowEnabled(hwndSysmenu))
        fSuccess = WinEnableWindow(hwndSysmenu, TRUE);
}
```

WinIsThreadActive - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Example Code](#)
- [Related Functions](#)
- [Glossary](#)

WinIsWindow

WinIsWindow - Syntax

This function determines if a window handle is valid.


```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB     hab; /* Anchor-block handle. */
HWND    hwnd; /* Window handle. */
BOOL     rc; /* Validity indicator. */

rc = WinIsWindow(hab, hwnd);
```

WinIsWindow Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinIsWindow Parameter - hwnd

hwnd (**HWND**) - input
Window handle.

WinIsWindow Return Value - rc

rc (**BOOL**) - returns
Validity indicator.

TRUE	Window handle is valid
FALSE	Window handle is invalid.

WinIsWindow - Parameters

hab (**HAB**) - input
Anchor-block handle.

hwnd (**HWND**) - input
Window handle.

rc (**BOOL**) - returns
Validity indicator.

TRUE

	Window handle is valid
FALSE	Window handle is invalid.

WinIsWindow - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinIsWindow - Related Functions

Related Functions

- [WinEnableWindow](#)
 - [WinIsThreadActive](#)
 - [WinIsWindow](#)
 - [WinIsWindowEnabled](#)
 - [WinQueryDesktopWindow](#)
 - [WinQueryObjectWindow](#)
 - [WinQueryWindowDC](#)
 - [WinQueryWindowProcess](#)
 - [WinQueryWindowRect](#)
 - [WinWindowFromDC](#)
 - [WinWindowFromID](#)
 - [WinWindowFromPoint](#)
-

WinIsWindow - Example Code

This example uses WinIsWindow to verify that the parent window is valid before querying and enabling the system menu window via WinIsWindowEnabled and WinEnableWindow.

```
#define INCL_WINWINDOWMGR    /* Window Manager Functions */
#define INCL_WINFRAMEMGR    /* Window Frame Functions */
#include <os2.h>

HAB    hab;                /* Anchor-block handle */
HWND    hwndSystemMenu;    /* System menu window */
HWND    hwnd;              /* Parent window */
BOOL    fSuccess;          /* Success indicator */

/* If the handle specifies a valid window, */
/* query the enabled status of the system menu. */
if (WinIsWindow(hab, hwnd))
{
    /* Obtain the handle for the system menu */
    hwndSystemMenu = WinWindowFromID(hwnd, FID_SYSTEMMENU);

    /* If the system menu is disabled, enable it */
    if (!WinIsWindowEnabled(hwndSystemMenu))
        fSuccess = WinEnableWindow(hwndSystemMenu, TRUE);
}
```

WinIsWindow - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinIsWindowEnabled

WinIsWindowEnabled - Syntax

This function returns the state (enabled/disabled) of a window.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd; /* Window handle. */
BOOL    rc;    /* Enabled-state indicator. */

rc = WinIsWindowEnabled(hwnd);
```

WinIsWindowEnabled Parameter - hwnd

hwnd ([HWND](#)) - input
Window handle.

WinIsWindowEnabled Return Value - rc

rc ([BOOL](#)) - returns
Enabled-state indicator.

TRUE

FALSE	Window is enabled.
	Window is disabled.

WinIsWindowEnabled - Parameters

hwnd ([HWND](#)) - input
Window handle.

rc ([BOOL](#)) - returns
Enabled-state indicator.

TRUE	Window is enabled.
FALSE	Window is disabled.

WinIsWindowEnabled - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinIsWindowEnabled - Related Functions

Related Functions

- [WinEnableWindow](#)
- [WinIsThreadActive](#)
- [WinIsWindow](#)
- [WinIsWindowEnabled](#)
- [WinQueryDesktopWindow](#)
- [WinQueryObjectWindow](#)
- [WinQueryWindowDC](#)
- [WinQueryWindowProcess](#)
- [WinQueryWindowRect](#)
- [WinWindowFromDC](#)
- [WinWindowFromID](#)
- [WinWindowFromPoint](#)

WinIsWindowEnabled - Example Code

This example uses `WinIsWindowEnabled` to check that the parent window is currently disabled before calling `WinEnableWindow` to enable the system menu window.

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
```

```

#define INCL_WINFRAMEMGR      /* Window Frame Functions      */
#include <os2.h>

HAB    hab;                  /* Anchor-block handle      */
HWND    hwndSysmenu;         /* System menu window       */
HWND    hwnd;                /* Parent window            */
BOOL    fSuccess;            /* Success indicator        */

/* Obtain the handle for the system menu */
hwndSysmenu = WinWindowFromID(hwnd,FID_SYSMENU);

/* If the system menu is disabled, enable it */
if (!WinIsWindowEnabled(hwndSysmenu))
    fSuccess = WinEnableWindow(hwndSysmenu, TRUE);

```

WinIsWindowEnabled - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinIsWindowShowing

WinIsWindowShowing - Syntax

This function determines whether any part of the window *hwnd* is physically visible.

```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND    hwnd; /* Window handle. */
BOOL    rc;    /* Showing state indicator. */

rc = WinIsWindowShowing(hwnd);

```

WinIsWindowShowing Parameter - hwnd

hwnd ([HWND](#)) - input

Window handle.

WinIsWindowShowing Return Value - rc

rc ([BOOL](#)) - returns
Showing state indicator.

TRUE

Some part of the window is displayed on the screen.

FALSE

The function returns FALSE in these situations:

- No part of the window is displayed on the screen
 - The window is disabled
 - The window update is disabled
-

WinIsWindowShowing - Parameters

hwnd ([HWND](#)) - input
Window handle.

rc ([BOOL](#)) - returns
Showing state indicator.

TRUE

Some part of the window is displayed on the screen.

FALSE

The function returns FALSE in these situations:

- No part of the window is displayed on the screen
 - The window is disabled
 - The window update is disabled
-

WinIsWindowShowing - Remarks

This function is useful for applications that constantly output new information. If value FALSE is returned (that is, no part of the window is physically visible), the application can choose not to redraw, since redrawing is not necessary.

If an application is using WinIsWindowShowing, it must issue the call every time it has new information that needs to be updated. If this is not done, invalid screen content could result. The alternative to this approach for a constantly-updating application that has new information is for it to invalidate its window and redraw within a [WinBeginPaint](#) - [WinEndPaint](#) sequence.

FALSE is returned if the PM session is not currently visible.

WinIsWindowShowing - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)

An invalid window handle was specified.

WinIsWindowShowing - Related Functions

Related Functions

- [WinBeginPaint](#)
 - [WinEnableWindowUpdate](#)
 - [WinEndPaint](#)
 - [WinExcludeUpdateRegion](#)
 - [WinGetClipPS](#)
 - [WinGetPS](#)
 - [WinGetScreenPS](#)
 - [WinInvalidateRect](#)
 - [WinInvalidateRegion](#)
 - [WinIsWindowShowing](#)
 - [WinIsWindowVisible](#)
 - [WinLockVisRegions](#)
 - [WinOpenWindowDC](#)
 - [WinQueryUpdateRect](#)
 - [WinQueryUpdateRegion](#)
 - [WinRealizePalette](#)
 - [WinReleasePS](#)
 - [WinShowWindow](#)
 - [WinUpdateWindow](#)
 - [WinValidateRect](#)
 - [WinValidateRegion](#)
-

WinIsWindowShowing - Example Code

This example uses WinIsWindowShowing to check if any part of the window is physically visible before causing a redraw of the window via WinInvalidateRect.

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#define INCL_WINFRAMEMGR      /* Window Frame Functions */
#include <os2.h>

HWND    hwnd;                /* Window handle */
RECT    rcl;                 /* Update region */

/* If any part of the window is visible, cause a redraw */
if (WinIsWindowShowing(hwnd))
{
    WinQueryUpdateRect(hwnd, &rcl);
    WinInvalidateRect(hwnd, /* Window to invalidate */
                      &rcl, /* Invalid rectangle */
                      FALSE); /* Do not include children */
}
```

WinIsWindowShowing - Topics

Select an item:

[Syntax](#)

[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinIsWindowVisible

WinIsWindowVisible - Syntax

This function returns the visibility state of a window.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd; /* Window handle. */
BOOL    rc;    /* Visibility-state indicator. */

rc = WinIsWindowVisible(hwnd);
```

WinIsWindowVisible Parameter - hwnd

hwnd (**HWND**) - input
Window handle.

WinIsWindowVisible Return Value - rc

rc (**BOOL**) - returns
Visibility-state indicator.

- | | |
|-------|---|
| TRUE | Window and all its parents have the WS_VISIBLE style bit set on |
| FALSE | Window or one of its parents have the WS_VISIBLE style bit set off. |

WinIsWindowVisible - Parameters

hwnd ([HWND](#)) - input
Window handle.

rc ([BOOL](#)) - returns
Visibility-state indicator.

TRUE	Window and all its parents have the WS_VISIBLE style bit set on
FALSE	Window or one of its parents have the WS_VISIBLE style bit set off.

WinIsWindowVisible - Remarks

Because *rc* reflects only the values of WS_VISIBLE style bits, *rc* may be set to TRUE even if *hwnd* is totally obscured by other windows. Use [WinIsWindowShowing](#) to determine if any part of the window is actually visible.

WinIsWindowVisible - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinIsWindowVisible - Related Functions

Related Functions

- [WinBeginPaint](#)
- [WinEnableWindowUpdate](#)
- [WinEndPaint](#)
- [WinExcludeUpdateRegion](#)
- [WinGetClipPS](#)
- [WinGetPS](#)
- [WinGetScreenPS](#)
- [WinInvalidateRect](#)
- [WinInvalidateRegion](#)
- [WinIsWindowShowing](#)
- [WinIsWindowVisible](#)
- [WinLockVisRegions](#)
- [WinOpenWindowDC](#)
- [WinQueryUpdateRect](#)
- [WinQueryUpdateRegion](#)
- [WinRealizePalette](#)
- [WinReleasePS](#)
- [WinShowWindow](#)
- [WinUpdateWindow](#)
- [WinValidateRect](#)
- [WinValidateRegion](#)

WinIsWindowVisible - Example Code

This example uses WinIsWindowVisible to query the visibility state of a window (i.e. the value of the WS_VISIBLE style bits) when the window is created, so that the window can be designated as visible, if necessary, by calling WinEnableWindowUpdate.

```
#define INCL_WINWINDOWMGR    /* Window Manager Functions */
#include <os2.h>

HWND    hwnd;                /* Parent window          */
BOOL    fSuccess;            /* Success indicator      */

case WM_CREATE:
    /* If the window has WS_VISIBLE off, set state to visible */
    if (!WinIsWindowVisible(hwnd))
    {
        /* Set the state to visible and cause the WM_PAINT message */
        fSuccess = WinEnableWindowUpdate(hwnd, TRUE);
    }
```

WinIsWindowVisible - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinIsWPDServerReady

WinIsWPDServerReady - Syntax

This function returns the state of the Workplace Shell DSOM server.

```
#define INCL_WPCCLASS
#include <os2.h>

BOOL    fReady; /* DSOM server-state indicator. DSOM Server status. */

fReady = WinIsWPDServerReady();
```

WinIsWPDServerReady Return Value - fReady

fReady ([BOOL](#)) - returns
DSOM server-state indicator. DSOM Server status.

TRUE	Workplace Shell DSOM Server is ready.
FALSE	Workplace Shell DSOM Server is not ready.

WinIsWPDServerReady - Parameters

fReady ([BOOL](#)) - returns
DSOM server-state indicator. DSOM Server status.

TRUE	Workplace Shell DSOM Server is ready.
FALSE	Workplace Shell DSOM Server is not ready.

WinIsWPDServerReady - Remarks

This function returns the ready status of the Workplace Shell DSOM Server.

Note: This function requires that the PM Shell is up and running.

WinIsWPDServerReady - Related Functions

Related Functions

- [WinIsSOMDDReady](#)
- [WinRestartSOMDD](#)
- [WinRestartWPDServer](#)

WinIsWPDServerReady - Example Code

This example stops the Workplace Shell DSOM Server, then waits until the server has terminate before stopping the DSOM daemon.

```
#define INCL_WPCCLASS
#include <os2.h>

ULONG count=0;
```

```

enum    {OFF, ON};

WinRestartWPDServer(OFF);

/* Make sure the server thread has terminated completely before */
/* bring down the DSOM daemon */
while ( WinIsWPDServerReady() )
{
    HEV hev;

    /* First create a private, reset, event semaphore. */
    DosCreateEventSem( (PSZ)NULL, &hev, 0, FALSE);

    /* Wait for 1 second; then try again for a max. of 30 sec. */
    DosWaitEventSem (hev, 1000);
    if (count++ > 30)
        break;
}
WinRestartSOMDD (OFF);

```

WinIsWPDServerReady - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinLoadAccelTable

WinLoadAccelTable - Syntax

This function loads an accelerator table.

```

#define INCL_WINACCELERATORS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;           /* Anchor-block handle. */
HMODULE   Resource;      /* Resource identity containing the accelerator table. */
ULONG     idAccelTable;  /* Accelerator-table identifier, within the resource file. */
HACCEL    haccelAccel;   /* Accelerator-table handle. */

haccelAccel = WinLoadAccelTable(hab, Resource,
                                idAccelTable);

```

WinLoadAccelTable Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinLoadAccelTable Parameter - Resource

Resource ([HMODULE](#)) - input
Resource identity containing the accelerator table.

Module handle returned by the DosLoadModule or DosQueryModuleHandle functions referencing a dynamic link library containing the resource or NULLHANDLE for the application's module.

WinLoadAccelTable Parameter - idAccelTable

idAccelTable ([ULONG](#)) - input
Accelerator-table identifier, within the resource file.

It must be greater or equal to 0 and less or equal to 0xFFFF.

WinLoadAccelTable Return Value - haccelAccel

haccelAccel ([HACCEL](#)) - returns
Accelerator-table handle.

WinLoadAccelTable - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

Resource ([HMODULE](#)) - input
Resource identity containing the accelerator table.

Module handle returned by the DosLoadModule or DosQueryModuleHandle functions referencing a dynamic link library containing the resource or NULLHANDLE for the application's module.

idAccelTable ([ULONG](#)) - input
Accelerator-table identifier, within the resource file.

It must be greater or equal to 0 and less or equal to 0xFFFF.

haccelAccel ([HACCEL](#)) - returns
Accelerator-table handle.

WinLoadAccelTable - Remarks

This function returns a different value when called twice in succession with the same parameter values.

The accelerator table is owned by the process from which this function is issued. It cannot be accessed directly from any other process. If it still exists when the process terminates, it is automatically deleted by the system.

WinLoadAccelTable - Errors

Possible returns from [WinGetLastError](#)

PMERR_RESOURCE_NOT_FOUND (0x100A)
The specified resource identity could not be found.

WinLoadAccelTable - Related Functions

Related Functions

- [WinCopyAccelTable](#)
- [WinCreateAccelTable](#)
- [WinDestroyAccelTable](#)
- [WinLoadAccelTable](#)
- [WinQueryAccelTable](#)
- [WinSetAccelTable](#)
- [WinTranslateAccel](#)

WinLoadAccelTable - Example Code

This example loads an accelerator-table, using the application defined accelerator id, from a resource using the resource handle returned by `DosLoadModule` or `DosQueryModuleHandle`. The returned table handle is then used by `WinCopyAccelTable` to copy the table into an in-memory accelerator table structure.

```
#define INCL_WINACCELERATORS    /* Window Accelerator Functions */
#define INCL_DOSMODULEMGR      /* Module Manager Functions      */
#include <os2.h>
#define ACCEL_ID 1

ULONG    ulCopied;              /* bytes copied                    */
HACCEL   hAccel;               /* Accelerator-table handle        */
ACCELTABLE pacctAccelTable; /* Accelerator-table data area    */
ULONG    ulCopyMax;            /* Maximum data area size         */
ULONG    idAccelTable=ACCEL_ID; /* Accelerator-table identifier   */
HAB      hab;                 /* anchor-block handle            */
HMODULE   hmodDLL;            /* resource module                */
CHAR      LoadError[100]; /* object name buffer for DosLoad */
```

```

ULONG    rc;                /* return code */

/* obtain resource handle */
rc = DosLoadModule(LoadError, sizeof(LoadError), "RES.DLL",
                  &hmodDLL);

if (rc == 0)
    hAccel = WinLoadAccelTable(hab, hmodDLL, idAccelTable);

ulCopyMax = sizeof(pacctAccelTable);
if (hAccel)
    ulCopied = WinCopyAccelTable(hAccel, &pacctAccelTable,
                                ulCopyMax);

```

WinLoadAccelTable - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinLoadDlg

WinLoadDlg - Syntax

This function creates a dialog window from the dialog template *idDlg* in *hmod* and returns the dialog window handle.

```

#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND      hwndParent;      /* Parent-window handle of the created dialog window. */
HWND      hwndOwner;       /* Requested owner-window handle of the created dialog window. */
PFNWNDP   pfnDlgProc;      /* Dialog procedure for the created dialog window. */
HMODULE    hmod;           /* Resource identity containing the dialog template. */
ULONG     idDlg;           /* Dialog-template identity within the resource file. */
PVOID     pCreateParams;   /* Pointer to application-defined data area. */
HWND      hwndDlg;         /* Dialog-window handle. */

hwndDlg = WinLoadDlg(hwndParent, hwndOwner,
                    pfnDlgProc, hmod, idDlg, pCreateParams);

```

WinLoadDlg Parameter - hwndParent

hwndParent ([HWND](#)) - input
Parent-window handle of the created dialog window.

HWND_DESKTOP	The desktop window
HWND_OBJECT	Object window
Other	Specified window.

WinLoadDlg Parameter - hwndOwner

hwndOwner ([HWND](#)) - input
Requested owner-window handle of the created dialog window.

The actual owner window is calculated using the algorithm specified below.

WinLoadDlg Parameter - pfnDlgProc

pfnDlgProc ([PFNWP](#)) - input
Dialog procedure for the created dialog window.

WinLoadDlg Parameter - hmod

hmod ([HMODULE](#)) - input
Resource identity containing the dialog template.

NULLHANDLE	Use the application's .EXE file.
Other	Module handle returned from the DosLoadModule or DosQueryModuleHandle functions.

WinLoadDlg Parameter - idDlg

idDlg ([ULONG](#)) - input
Dialog-template identity within the resource file.

It is also used as the identity of the created dialog window.

WinLoadDlg Parameter - pCreateParams

pCreateParams ([PVOID](#)) - input

Pointer to application-defined data area.

This is passed to the dialog procedure in the [WM_INITDLG](#) message.

This parameter MUST be a pointer rather than a long.

WinLoadDlg Return Value - hwndDlg

hwndDlg ([HWND](#)) - returns

Dialog-window handle.

NULL

Dialog window not created

Other

Dialog window handle.

WinLoadDlg - Parameters

hwndParent ([HWND](#)) - input

Parent-window handle of the created dialog window.

HWND_DESKTOP

The desktop window

HWND_OBJECT

Object window

Other

Specified window.

hwndOwner ([HWND](#)) - input

Requested owner-window handle of the created dialog window.

The actual owner window is calculated using the algorithm specified below.

pfnDlgProc ([PFNWP](#)) - input

Dialog procedure for the created dialog window.

hmod ([HMODULE](#)) - input

Resource identity containing the dialog template.

NULLHANDLE

Use the application's .EXE file.

Other

Module handle returned from the DosLoadModule or DosQueryModuleHandle functions.

idDlg ([ULONG](#)) - input

Dialog-template identity within the resource file.

It is also used as the identity of the created dialog window.

pCreateParams ([PVOID](#)) - input

Pointer to application-defined data area.

This is passed to the dialog procedure in the [WM_INITDLG](#) message.

This parameter **MUST** be a pointer rather than a long.

hwndDlg (HWND) - returns
Dialog-window handle.

NULL	Dialog window not created
Other	Dialog window handle.

WinLoadDlg - Remarks

Unless window style `WS_VISIBLE` is specified for the dialog window in the `DIALOG` statement within the dialog template, the dialog window is created as an invisible window.

The dialog window owner may be modified, in order to ensure acceptable results if it is later processed as a modal dialog using the [WinProcessDlg](#) or [WinGetDlgMsg](#) functions. A search is made up the parent hierarchy, starting at the window specified by the *hwndOwner* parameter, until a child of the window specified by the *hwndParent* is found. If such a window exists, it is made the actual owner of the dialog. If no such window exists the actual owner of the dialog is set to `NULLHANDLE`.

This function returns immediately after creating the dialog window. A [WM_INITDLG \(Default Dialogs\)](#) message is sent to the dialog procedure before this function returns.

This function should not be used while pointing device capture is set (see [WinSetCapture](#)).

As each of the controls defined within the template of this dialog window is created during the processing of this function, the dialog procedure may receive various control notifications before this function returns.

A dialog window can be destroyed with the [WinDestroyWindow](#) function.

Because windows are created from the template, strings in the template are processed with [WinSubstituteStrings](#). Any resultant [WM_SUBSTITUTESTRING](#) messages are sent to the dialog procedure before this function returns.

When the child windows of the dialog are created, the [WinSubstituteStrings](#) function is used to allow the child windows to perform text substitutions in their window text. If any of the child window text strings contain the percent (%) substitution character, there is an upper limit of 256 on the length of the text string, after it is returned from the substitution.

If a dialog template (typically compiled using the resource compiler) references another resource (for example an icon resource for an icon static control), this function always searches for that resource in the .EXE file. If an application wishes to keep resources referenced by a dialog template in a .DLL library, these resources must be loaded by an explicit function call during the processing of the [WM_INITDLG](#) message.

Note: In general, it is better to create the dialog window invisible as this allows for optimization. In particular, an experienced user can type ahead, anticipating the processing in the dialog window.

In this instance, there may be no need to display the dialog window at all, as the user might have finished the interaction before the window can be displayed.

This is in fact how the [WinProcessDlg](#) function works; it does not display the dialog window while there are still [WM_CHAR](#) messages in the input queue, but allows these to be processed first.

WinLoadDlg - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_INVALID_INTEGER_ATOM (0x1016)
The specified atom is not a valid integer atom.

PMERR_INVALID_ATOM_NAME (0x1015)
An invalid atom name string was passed.

PMERR_ATOM_NAME_NOT_FOUND (0x1017)
The specified atom name is not in the atom table.

PMERR_RESOURCE_NOT_FOUND (0x100A)
The specified resource identity could not be found.

WinLoadDlg - Related Functions

Related Functions

- [WinCreateDlg](#)
 - [WinDefDlgProc](#)
 - [WinDismissDlg](#)
 - [WinDlgBox](#)
 - [WinGetDlgMsg](#)
 - [WinLoadDlg](#)
 - [WinProcessDlg](#)
-

WinLoadDlg - Related Messages

Related Messages

- [WM_INITDLG](#) (Default Dialogs)
 - [WM_SUBSTITUTESTRING](#)
 - [WM_CHAR](#)
-

WinLoadDlg - Example Code

This example uses WinLoadDlg to load a dialog template from the application's .EXE file.

```
#define INCL_WINDIALOGS      /* Window Dialog Mgr Functions */
#include <os2.h>

HWND  hwndOwner;           /* owner window */
HWND  hwndDlg;             /* Dialog-window handle */
PFNWP GenericDlgProc;      /* dialog process */

hwndDlg = WinLoadDlg(hwndDesktop, /* parent is desk top */
                    hwndOwner,    /* owner is main frame */
                    GenericDlgProc, /* dialog procedure */
                    0L,           /* load from resource file */
                    DLG_ID,       /* dialog resource id */
                    NULL);        /* no dialog parameters */
```

WinLoadDlg - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinLoadFileIcon

WinLoadFileIcon - Syntax

This function returns a pointer to an icon which is associated with the file specified by *pszFileName*.

```
#define INCL_WINWORKPLACE
#include <os2.h>

PSZ      pszFileName; /* Pointer to file name. */
BOOL     fPrivate;    /* Icon usage flag. */
HPOINTER hIcon;       /* Icon handle. */

hIcon = WinLoadFileIcon(pszFileName, fPrivate);
```

WinLoadFileIcon Parameter - pszFileName

pszFileName ([PSZ](#)) - input
Pointer to file name.

A pointer to a zero-terminated string which contains the name of the file whose icon will be loaded.

WinLoadFileIcon Parameter - fPrivate

fPrivate ([BOOL](#)) - input

Icon usage flag.

TRUE

A private copy of this icon is requested. This flag should be used if the application needs to modify the icon.

FALSE

A shared pointer to this icon is requested. This flag should be used if application needs to display the icon without modifying it. This should be used whenever possible to optimize system resource use.

WinLoadFileIcon Return Value - hIcon

hIcon (**HPOINTER**) - returns
Icon handle.

NULL

Error occurred.

OTHER

Handle to an icon.

WinLoadFileIcon - Parameters

pszFileName (**PSZ**) - input
Pointer to file name.

A pointer to a zero-terminated string which contains the name of the file whose icon will be loaded.

fPrivate (**BOOL**) - input
Icon usage flag.

TRUE

A private copy of this icon is requested. This flag should be used if the application needs to modify the icon.

FALSE

A shared pointer to this icon is requested. This flag should be used if application needs to display the icon without modifying it. This should be used whenever possible to optimize system resource use.

hIcon (**HPOINTER**) - returns
Icon handle.

NULL

Error occurred.

OTHER

Handle to an icon.

WinLoadFileIcon - Remarks

The icon will be retrieved in the following order until an icon has been found:

- .ICON extended attribute
- .ICO file in same directory with same prefix
- Application specific icon (if PM executable or MS Windows* executable)

- PM application icon (if PM executable)
- MS Windows* application icon (if MS Windows* application executable)
- OS/2 application icon (if OS/2 full-screen only executable)
- OS/2 window icon (if OS/2 window compatible executable)
- DOS windowed application icon (if DOS windowed executable)
- Program application (if unknown type executable)
- Data icon specified by associated application
- Data icon of associated application
- Data file icon (if not program or directory)
- Directory icon (if directory)

The [HPOINTER](#) returned in *Private* should be freed by the caller via [WinFreeFileIcon](#) when it is no longer being used.

WinLoadFileIcon - Related Functions

Related Functions

- [WinSetFileIcon](#)
- [WinFreeFileIcon](#)

WinLoadFileIcon - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

WinLoadHelpTable

WinLoadHelpTable - Syntax

This function identifies the module handle and identity of the help table to the instance of the Help Manager.

```
#define INCL_WINHELP /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndHelpInstance; /* Handle of an instance of the Help Manager. */
ULONG      idHelpTable;     /* Identity of the help table. */
HMODULE     Module;         /* Handle of the module which contains the help table and help subtable resource. */
BOOL       rc;              /* Success indicator. */

rc = WinLoadHelpTable(hwndHelpInstance, idHelpTable,
    Module);
```

WinLoadHelpTable Parameter - hwndHelpInstance

hwndHelpInstance ([HWND](#)) - input
Handle of an instance of the Help Manager.

This is the handle returned by the [WinCreateHelpInstance](#) call.

WinLoadHelpTable Parameter - idHelpTable

idHelpTable ([ULONG](#)) - input
Identity of the help table.

It must be greater or equal to 0 and less or equal to 0xFFFF.

WinLoadHelpTable Parameter - Module

Module ([HMODULE](#)) - input
Handle of the module which contains the help table and help subtable resources.

- NULLHANDLE Use the resources file for the application.
- Other The module handle returned by the DosLoadModule or DosQueryModuleHandle call referencing a dynamic-link library containing the help resources.

WinLoadHelpTable Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

- TRUE Successful completion
- FALSE Error occurred.

WinLoadHelpTable - Parameters

hwndHelpInstance ([HWND](#)) - input
Handle of an instance of the Help Manager.

This is the handle returned by the [WinCreateHelpInstance](#) call.

idHelpTable ([ULONG](#)) - input
Identity of the help table.

It must be greater or equal to 0 and less or equal to 0xFFFF.

Module ([HMODULE](#)) - input
Handle of the module which contains the help table and help subtable resources.

NULLHANDLE	Use the resources file for the application.
Other	The module handle returned by the DosLoadModule or DosQueryModuleHandle call referencing a dynamic-link library containing the help resources.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinLoadHelpTable - Remarks

An application specifies or changes the handle of the module which contains the help table or the identity of the help table.

This function corresponds to the [HM_LOAD_HELP_TABLE](#) message that identifies the identifier of a help table and the handle of the module which contains the help table and its associated help subtables.

WinLoadHelpTable - Related Functions

- Related Functions**
- [WinAssociateHelpInstance](#)
 - [WinCreateHelpInstance](#)
 - [WinCreateHelpTable](#)
 - [WinDestroyHelpInstance](#)
 - [WinLoadHelpTable](#)
 - [WinQueryHelpInstance](#)

WinLoadHelpTable - Related Messages

- Related Messages**
- [HM_LOAD_HELP_TABLE](#)

WinLoadHelpTable - Example Code

```
BOOL LoadHelpTable( HWND hWnd, USHORT usResource, PSZ pszModuleName )
{
    BOOL bSuccess = FALSE;
    HMODULE hmodule;
    HWND hwndHelp;
    PSZ pszObjNameBuf[ 80 ];

    /* Get the DLL loaded */
    if( !DosLoadModule( pszObjNameBuf, sizeof( pszObjNameBuf ),
                      pszModuleName, &hmodule ) )
    {
        /* Get the associated help instance */
        hwndHelp = WinQueryHelpInstance( hWnd );

        if( hwndHelp )
        {
            /* Pass address of help table to the Help Manager */
            bSuccess = WinLoadHelpTable( hwndHelp, usResource, hmodule );
        }
    }

    /* Return success indicator */
    return bSuccess;
}
```

WinLoadHelpTable - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinLoadLibrary

WinLoadLibrary - Syntax

This function makes the library available to the application.

```
#define INCL_WINLOAD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
```

```
HAB      hab;           /* Anchor-block handle. */
PSZ      pszLibname;    /* Library name. */
HLIB     hlibLibhandle; /* Library handle. */

hlibLibhandle = WinLoadLibrary(hab, pszLibname);
```

WinLoadLibrary Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinLoadLibrary Parameter - pszLibname

pszLibname ([PSZ](#)) - input
Library name.

WinLoadLibrary Return Value - hlibLibhandle

hlibLibhandle ([HLIB](#)) - returns
Library handle.

NULLHANDLE	Library not successfully loaded
Other	Library handle.

WinLoadLibrary - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

pszLibname ([PSZ](#)) - input
Library name.

hlibLibhandle ([HLIB](#)) - returns
Library handle.

NULLHANDLE	Library not successfully loaded
Other	Library handle.

WinLoadLibrary - Remarks

This function makes the library *pszLibname* (containing procedures, or resources, or both) available to the application. All of the dynamic link libraries have the .DLL filename extension by default.

WinLoadLibrary - Related Functions

Related Functions

- [WinDeleteLibrary](#)
 - [WinDeleteProcedure](#)
 - [WinLoadLibrary](#)
 - [WinLoadProcedure](#)
-

WinLoadLibrary - Example Code

This example loads the RES.DLL resource/procedure library, returning a library handle that is then used by WinLoadProcedure to load procedures from that library.

```
#define INCL_WINLOAD          /* Window Load Functions          */
#include <os2.h>

PFNWP  pWndproc;             /* procedure pointer          */
HAB     hab;                 /* anchor-block handle        */
HLIB     hlib;               /* library handle             */
char     pszLibname[10]="RES.DLL"; /* library name string        */
char     pszProcname[10]="WndProc"; /* procedure name string      */

/* load RES.DLL */
hlib = WinLoadLibrary(hab, pszLibname);

/* load WndProc */
pWndproc = WinLoadProcedure(hab, hlib, pszProcname);
```

WinLoadLibrary - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinLoadMenu

WinLoadMenu - Syntax

This function creates a menu window from the menu template *idMenu* from *hmod*, and returns in *hwndMenu* the window handle for the created window.

```
#define INCL_WINMENUS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndFrame; /* Owner- and parent-window handle. */
HMODULE    hmod;      /* Resource identifier. */
ULONG      idMenu;    /* Menu identifier within the resource file. */
HWND      hwndMenu;   /* Menu-window handle. */

hwndMenu = WinLoadMenu(hwndFrame, hmod, idMenu);
```

WinLoadMenu Parameter - hwndFrame

hwndFrame (**HWND**) - input
Owner- and parent-window handle.

HWND_DESKTOP	The desktop window
HWND_OBJECT	Object window
Other	Specified window.

WinLoadMenu Parameter - hmod

hmod (**HMODULE**) - input
Resource identifier.

NULLHANDLE	The resource is in the .EXE file of the application.
Other	The module handle returned by the DosLoadModule or DosQueryModuleHandle call.

WinLoadMenu Parameter - idMenu

idMenu (**ULONG**) - input

Menu identifier within the resource file.

It must be greater or equal to 0 and less or equal to 0xFFFF.

WinLoadMenu Return Value - hwndMenu

hwndMenu ([HWND](#)) - returns
Menu-window handle.

WinLoadMenu - Parameters

hwndFrame ([HWND](#)) - input
Owner- and parent-window handle.

HWND_DESKTOP	The desktop window
HWND_OBJECT	Object window
Other	Specified window.

hmod ([HMODULE](#)) - input
Resource identifier.

NULLHANDLE	The resource is in the .EXE file of the application.
Other	The module handle returned by the DosLoadModule or DosQueryModuleHandle call.

idMenu ([ULONG](#)) - input
Menu identifier within the resource file.

It must be greater or equal to 0 and less or equal to 0xFFFF.

hwndMenu ([HWND](#)) - returns
Menu-window handle.

WinLoadMenu - Remarks

The menu window is created with its parent and owner set to *hwndFrame*, and with identity FID_MENU. If *hwndFrame* is HWND_OBJECT or a window handle returned from [WinQueryObjectWindow](#), the menu window is created as an object window.

Action bar menus are created as child windows of the frame window and are initially visible. Submenus are initially created as object windows that are owned by the window frame.

WinLoadMenu - Related Functions

Related Functions

- [WinCreateMenu](#)
- [WinLoadMenu](#)
- [WinPopupMenu](#)

WinLoadMenu - Example Code

This example creates a menu window from the menu template (idMenuId) located in "RES.DLL" and returns a menu handle which is used by WinPopupMenu.

```
#define INCL_WINWINDOWMGR /* Window Manager Functions */
#define INCL_WINMENUS /* Window Menu Functions */
#include <os2.h>

HWND hwndMenu; /* Menu window */
HWND hwndOwner; /* Owner window */
HMODULE hmodDLL; /* resource handle */
ULONG idMenuId; /* Resource menu id */
BOOL fSuccess; /* Success indicator */
HWND hwndParent; /* Parent window */
ULONG flOptions; /* Pop-up menu options */

if (DosQueryModuleHandle("RES.DLL",&hmodDLL))
    hwndMenu = WinLoadMenu(hwndOwner, hmodDLL, idMenuId);

flOptions = PU_MOUSEBUTTON1DOWN |
            PU_KEYBOARD |
            PU_MOUSEBUTTON1;
fSuccess = WinPopupMenu(hwndParent,
                        hwndOwner,
                        hwndMenu,
                        0, 50, 0,
                        flOptions);
```

WinLoadMenu - Topics

Select an item:

- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Example Code](#)
 - [Related Functions](#)
 - [Glossary](#)
-

WinLoadMessage

WinLoadMessage - Syntax

This function loads a message from a resource, copies the message to the specified buffer, and appends a terminating null character.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;      /* Anchor-block handle. */
HMODULE  hmodMod;   /* Module handle. */
ULONG    ulId;      /* Message identifier. */
LONG     lcchMax;    /* Specifies the size of buffer. */
PSZ      pBuffer;   /* Points to the buffer that receives the message. */
LONG     lLength;    /* The length of the string returned. */

lLength = WinLoadMessage(hab, hmodMod, ulId,
                        lcchMax, pBuffer);
```

WinLoadMessage Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinLoadMessage Parameter - hmodMod

hmodMod (**HMODULE**) - input
Module handle.

NULLHANDLE
Use the application's own resources file.

Other
Module handle returned by the DosLoadModule or DosQueryModuleHandle call referencing a dynamic-link library containing the resource.

WinLoadMessage Parameter - ulId

ulId (**ULONG**) - input
Message identifier.

It must be greater or equal to 0 and less or equal to 0xFFFF.

WinLoadMessage Parameter - lcchMax

lcchMax (**LONG**) - input
Specifies the size of buffer.

The maximum length of a string is 256 character.

WinLoadMessage Parameter - pBuffer

pBuffer (**PSZ**) - input
Points to the buffer that receives the message.

WinLoadMessage Return Value - ILength

ILength (**LONG**) - returns
The length of the string returned.

This excludes the terminating null, and has the following values:

0	Error
Other	A maximum value of (<i>lcchMax</i> -1).

WinLoadMessage - Parameters

hab (**HAB**) - input
Anchor-block handle.

hmodMod (**HMODULE**) - input
Module handle.

NULLHANDLE	Use the application's own resources file.
Other	Module handle returned by the DosLoadModule or DosQueryModuleHandle call referencing a dynamic-link library containing the resource.

uId (**ULONG**) - input
Message identifier.

It must be greater or equal to 0 and less or equal to 0xFFFF.

lcchMax (**LONG**) - input
Specifies the size of buffer.

The maximum length of a string is 256 character.

pBuffer (**PSZ**) - input
Points to the buffer that receives the message.

ILength (**LONG**) - returns

The length of the string returned.

This excludes the terminating null, and has the following values:

0	Error
Other	A maximum value of (<i>lcchMax</i> -1).

WinLoadMessage - Remarks

Message resources contain up to 16 messages each. The resource ID is calculated from the id parameter value passed to this function as follows:

```
resource ID = (id / 16) + 1
```

To save storage on disk and in memory, applications should number their message resources sequentially, starting at some multiple of 16.

WinLoadMessage - Related Functions

Related Functions

- [WinLoadString](#)

WinLoadMessage - Example Code

This example loads an error message from ERR.DLL using the resource handle from DosLoadModule and uses the message in a message box.

```
#define INCL_WINWINDOWMGR          /* Window Manager Functions */
#define INCL_DOSMODULEMGR          /* Module Manager Functions */
#define INCL_WINDIALOGS           /* Window Dialog Mgr Functions */
#include <os2.h>
#define ERRMSG_ID 1

LONG    lLength;          /* length of string */
HAB     hab;              /* anchor-block handle */
HMODULE hmodDLL;          /* Handle of resource module */
LONG    lBufferMax = 100; /* Size of buffer */
char     pszErrMsg[100]; /* error message */
CHAR     LoadError[100]; /* object name buffer for DosLoad */
ULONG    rc;              /* return code */
HWND     hwnd;            /* window handle */

/* obtain resource handle */
rc = DosLoadModule(LoadError, sizeof(LoadError), "ERR.DLL",
                  &hmodDLL);

/* load message from resource */
if (rc == 0)
{
    /* load error message string */
    lLength = WinLoadMessage(hab, hmodDLL, ERRMSG_ID, lBufferMax,
                          pszErrMsg);
}
```

```

/* display error message box */
WinMessageBox(HWND_DESKTOP,
    hwnd,                /* client-window handle */
    pszErrMsg,           /* message */
    "Error message",     /* title of the message */
    0,                   /* message box id */
    MB_NOICON | MB_OK); /* icon and button flags */
}

```

WinLoadMessage - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinLoadPointer

WinLoadPointer - Syntax

This function loads a pointer from a resource file into the system.

```

#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndDesktop; /* Desktop-window handle. */
HMODULE    Resource;    /* Resource identity containing the pointer definition. */
ULONG      idPointer;   /* Identifier of the pointer to be loaded. */
HPOINTER   hptr;        /* Pointer handle. */

hptr = WinLoadPointer(hwndDesktop, Resource,
    idPointer);

```

WinLoadPointer Parameter - hwndDesktop

hwndDesktop (HWND) - input
Desktop-window handle.

HWND_DESKTOP

Other	The desktop window
	Desktop-window handle returned by WinQueryDesktopWindow .

WinLoadPointer Parameter - Resource

Resource ([HMODULE](#)) - input
Resource identity containing the pointer definition.

NULLHANDLE	Use the resources file for the application.
------------	---

Other	Module handle returned by the DosLoadModule or DosQueryModuleHandle call referencing a dynamic-link library containing the resource.
-------	--

WinLoadPointer Parameter - idPointer

idPointer ([ULONG](#)) - input
Identifier of the pointer to be loaded.

It must be greater or equal to 0 and less or equal to 0xFFFF.

WinLoadPointer Return Value - hptr

hptr ([HPOINTER](#)) - returns
Pointer handle.

NULLHANDLE	Error has occurred
------------	--------------------

Other	Handle of loaded pointer.
-------	---------------------------

WinLoadPointer - Parameters

hwndDeskTop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP	The desktop window
--------------	--------------------

Other	Desktop-window handle returned by WinQueryDesktopWindow .
-------	---

Resource ([HMODULE](#)) - input

Resource identity containing the pointer definition.

NULLHANDLE

Use the resources file for the application.

Other

Module handle returned by the `DosLoadModule` or `DosQueryModuleHandle` call referencing a dynamic-link library containing the resource.

idPointer ([ULONG](#)) - input

Identifier of the pointer to be loaded.

It must be greater or equal to 0 and less or equal to 0xFFFF.

hptr ([HPOINTER](#)) - returns

Pointer handle.

NULLHANDLE

Error has occurred

Other

Handle of loaded pointer.

WinLoadPointer - Remarks

A new copy of the pointer is created each time this function is called. The pointer created by this function can be destroyed using the [WinDestroyPointer](#) function. To get one of the standard system pointers, use the [WinQuerySysPointer](#) function.

The pointer is owned by the process from which this function is issued. It cannot be accessed directly from any other process. If it still exists when the process terminates, it is automatically deleted by the system.

WinLoadPointer - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)

An invalid window handle was specified.

PMERR_RESOURCE_NOT_FOUND (0x100A)

The specified resource identity could not be found.

WinLoadPointer - Related Functions

Related Functions

- [WinCreatePointer](#)
- [WinCreatePointerIndirect](#)
- [WinDestroyPointer](#)
- [WinDrawPointer](#)
- [WinLoadPointer](#)
- [WinQueryPointer](#)
- [WinQueryPointerInfo](#)
- [WinQueryPointerPos](#)
- [WinQuerySysPointer](#)
- [WinQuerySysPointerData](#)
- [WinSetPointer](#)

- [WinSetPointerPos](#)
- [WinSetSysPointerData](#)
- [WinShowPointer](#)

WinLoadPointer - Example Code

This example calls WinLoadPointer to load an application defined pointer. When processing the WM_MOUSEMOVE message, the loaded pointer is displayed by calling WinSetPointer.

```
#define INCL_WINPOINTERS          /* Window Pointer Functions */
#include <os2.h>

HPOINTER  hptrCrossHair; /* pointer handle */

case WM_CREATE:
    hptrCrossHair = WinLoadPointer(HWND_DESKTOP,
        0L,          /* load from .exe file */
        IDP_CROSSHAIR); /* identifies the pointer */

case WM_MOUSEMOVE:
    WinSetPointer(HWND_DESKTOP, hptrCrossHair);
```

WinLoadPointer - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinLoadProcedure

WinLoadProcedure - Syntax

This function loads the window or dialog procedure from a specified dynamic link library.

```
#define INCL_WINLOAD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
HLIB     hlibLibhandle; /* Library handle. */
PSZ      pszProcname;  /* Procedure name. */
```

```
PFNWP Wndproc; /* Window-procedure identifier. */  
  
Wndproc = WinLoadProcedure(hab, hlibLibhandle,  
                           pszProcname);
```

WinLoadProcedure Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinLoadProcedure Parameter - hlibLibhandle

hlibLibhandle ([HLIB](#)) - input
Library handle.

WinLoadProcedure Parameter - pszProcname

pszProcname ([PSZ](#)) - input
Procedure name.

WinLoadProcedure Return Value - Wndproc

Wndproc ([PFNWP](#)) - returns
Window-procedure identifier.

NULL	Procedure not successfully loaded
Other	Window-procedure identifier.

WinLoadProcedure - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

hlibLibhandle ([HLIB](#)) - input

Library handle.

pszProcname ([PSZ](#)) - input
Procedure name.

Wndproc ([PFNWP](#)) - returns
Window-procedure identifier.

NULL	Procedure not successfully loaded
Other	Window-procedure identifier.

WinLoadProcedure - Remarks

This function loads the window or dialog procedure *pszProcname* from the library *hlibLibhandle*.

WinLoadProcedure - Related Functions

Related Functions

- [WinDeleteLibrary](#)
 - [WinDeleteProcedure](#)
 - [WinLoadLibrary](#)
 - WinLoadProcedure
-

WinLoadProcedure - Example Code

This example loads the WndProc procedure, returning a pointer to the procedure, from the RES.DLL library, based on the library handle returned by WinLoadLibrary.

```
#define INCL_WINLOAD          /* Window Load Functions          */
#include <os2.h>

PFNWP  pWndproc;             /* procedure pointer          */
HAB     hab;                 /* anchor-block handle        */
HLIB     hlib;               /* library handle             */
char     pszLibname[10]="RES.DLL"; /* library name string        */
char     pszProcname[10]="WndProc"; /* procedure name string      */

/* load RES.DLL */
hlib = WinLoadLibrary(hab, pszLibname);

/* load WndProc */
pWndproc = WinLoadProcedure(hab, hlib, pszProcname);
```

WinLoadProcedure - Topics

Select an item:

- Syntax
- Parameters
- Returns
- Remarks
- Example Code
- Related Functions
- Glossary

WinLoadString

WinLoadString - Syntax

This function loads a string from a resource.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
HMODULE   Resource;     /* Resource identity containing the string. */
ULONG     idString;     /* String identifier. */
LONG      lBufferMax;   /* Size of buffer. */
PSZ       pszBuffer;    /* Buffer that is to receive the string. */
LONG      lLength;      /* The length of the string returned. */

lLength = WinLoadString(hab, Resource, idString,
                        lBufferMax, pszBuffer);
```

WinLoadString Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinLoadString Parameter - Resource

- Resource** (**HMODULE**) - input
Resource identity containing the string.
- NULLHANDLE**
Use the application's own resources file.
 - Other**
Module handle returned by the DosLoadModule or DosQueryModuleHandle call referencing a dynamic-link library containing the resource.

WinLoadString Parameter - idString

idString (ULONG) - input
String identifier.

It must be greater or equal to 0 and less or equal to 0xFFFF.

WinLoadString Parameter - IBufferMax

IBufferMax (LONG) - input
Size of buffer.

The maximum length of a string is 256 characters.

WinLoadString Parameter - pszBuffer

pszBuffer (PSZ) - output
Buffer that is to receive the string.

WinLoadString Return Value - ILength

ILength (LONG) - returns
The length of the string returned.

This excludes the terminating null, and has the following values:

- 0
 - Other Error
 - A maximum value of (*IBufferMax*-1).
-

WinLoadString - Parameters

hab (HAB) - input
Anchor-block handle.

Resource (HMODULE) - input

Resource identity containing the string.

NULLHANDLE

Use the application's own resources file.

Other

Module handle returned by the `DosLoadModule` or `DosQueryModuleHandle` call referencing a dynamic-link library containing the resource.

idString ([ULONG](#)) - input
String identifier.

It must be greater or equal to 0 and less or equal to 0xFFFF.

IBufferMax ([LONG](#)) - input
Size of buffer.

The maximum length of a string is 256 characters.

pszBuffer ([PSZ](#)) - output
Buffer that is to receive the string.

lLength ([LONG](#)) - returns
The length of the string returned.

This excludes the terminating null, and has the following values:

0

Error

Other

A maximum value of $(IBufferMax-1)$.

WinLoadString - Remarks

This function loads a string resource identified by the *idString* and the *Resource* parameters into the *pszBuffer* parameter, and appends a terminating null character.

RT_STRING resources (string resources) contain up to 16 strings each. The resource ID is calculated from the *idString* passed to this function as follows:

```
resource ID = (idString / 16) + 1
```

To save storage on disk and in memory, applications should number their string resources sequentially, starting at some multiple of 16.

WinLoadString - Errors

Possible returns from [WinGetLastError](#)

PMERR_RESOURCE_NOT_FOUND (0x100A)
The specified resource identity could not be found.

WinLoadString - Related Functions

Related Functions

- [WinCompareStrings](#)
- [WinLoadString](#)
- [WinNextChar](#)
- [WinPrevChar](#)
- [WinSubstituteStrings](#)
- [WinUpper](#)
- [WinUpperChar](#)

WinLoadString - Example Code

This example loads a string from RES.DLL using the resource handle from DosLoadModule.

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#define INCL_DOSMODULEMGR     /* Module Manager Functions */
#include <os2.h>
#define STRING_ID 1

LONG    lLength;           /* length of string */
HAB     hab;               /* anchor-block handle */
HMODULE hmodDLL;           /* Handle of resource module */
ULONG   idString = STRING_ID; /* String identifier */
LONG     lBufferMax = 10; /* Size of buffer */
char     pszString1[10]; /* first string */
CHAR     LoadError[100]; /* object name buffer for DosLoad */
ULONG    rc;               /* return code */

/* obtain resource handle */
rc = DosLoadModule(LoadError, sizeof(LoadError), "RES.DLL",
                  &hmodDLL);

/* load string from resource */
if (rc == 0)
    lLength = WinLoadString(hab, hmodDLL, idString, lBufferMax,
                          pszString1);
```

WinLoadString - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinLockPointerUpdate

WinLockPointerUpdate - Syntax

This function is specific to OS/2 Version 2.1 or higher.

This function causes the mouse pointer to change into the symbol described by *hptrNew* for the period of time indicated by *ulTimeInterval*.

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndDesktop; /* Handle to the desktop window. */
HPOINTER   hptrNew;     /* Pointer handle to be displayed. */
ULONG      ulTimeInterval; /* Time interval. */
BOOL       rc;          /* Success indicator. */

rc = WinLockPointerUpdate(hwndDesktop, hptrNew,
                          ulTimeInterval);
```

WinLockPointerUpdate Parameter - hwndDesktop

hwndDesktop (**HWND**) - input
Handle to the desktop window.

WinLockPointerUpdate Parameter - hptrNew

hptrNew (**HPOINTER**) - input
Pointer handle to be displayed.

WinLockPointerUpdate Parameter - ulTimeInterval

ulTimeInterval (**ULONG**) - input
Time interval.

The time (0 to 65,535ms) during which changes to the mouse pointer shape will be locked out.

WinLockPointerUpdate Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful.
FALSE	An error occurred.

WinLockPointerUpdate - Parameters

hwndDesktop ([HWND](#)) - input
Handle to the desktop window.

hptrNew ([HPOINTER](#)) - input
Pointer handle to be displayed.

ulTimeInterval ([ULONG](#)) - input
Time interval.

The time (0 to 65,535ms) during which changes to the mouse pointer shape will be locked out.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful.
FALSE	An error occurred.

WinLockPointerUpdate - Remarks

The mouse remains fully functional during the lock time, however the pointer shape is not updated.

This function can be used to convey visual feedback to the user, that some mouse action has been recognized. For example, that a gesture has been recognized from the last few mouse movements.

WinLockPointerUpdate - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

WinLockupSystem

WinLockupSystem - Syntax

This function is specific to OS/2 Version 2.1 or higher.

This function locks up the system.

```
#define INCL_WINMESSAGEGR
#include <os2.h>

HAB hab; /* The application anchor block. */
BOOL rc; /* Return code. */

rc = WinLockupSystem(hab);
```

WinLockupSystem Parameter - hab

hab (**HAB**) - input
The application anchor block.

WinLockupSystem Return Value - rc

rc (**BOOL**) - returns
Return code.

TRUE	The system was successfully locked.
FALSE	An error occurred or the system was already in a locked state.

WinLockupSystem - Parameters

hab (**HAB**) - input
The application anchor block.

rc (**BOOL**) - returns
Return code.

TRUE	The system was successfully locked.
FALSE	An error occurred or the system was already in a locked state.

WinLockupSystem - Remarks

This function allows an application program to cause the system to lock up at any point in time. For example, a programmer might wish to lock up the system if any tampering is detected, such as a password being misspelled more than four times in a row.

In order to execute [WinUnlockSystem](#) after a WinLockupSystem has been called, the [WinUnlockSystem](#) must be called from a separate thread because WinLockupSystem will not return until a password has been entered from the keyboard.

The [LockupHook](#) hook allows a PM application to customize system lockups.

In order for the window to appear as the top most window on the lockup screen, the WS_CLIPSIBLINGS flag must be used for the *//Style* parameter in the [WinCreateWindow](#) or [WinCreateStdWindow](#) call.

WinLockupSystem - Related Functions

Related Functions

- [LockupHook](#)
- [WinUnlockSystem](#)

WinLockupSystem - Example Code

```
HAB    hab          = NULLHANDLE;
HMQ    hmq          = NULLHANDLE;
QMSG   qmsg         = {0};
HWND   hwndFrame    = NULLHANDLE;
HWND   hwndClient   = NULLHANDLE;
CHAR   *szAppName   = "Lockup  ";
ULONG  FrameFlags    = FCF_STANDARD;
PSZ    pszPassword  = "m4a3x28t";

hab = WinInitialize( 0 );
hmq = WinCreateMsgQueue( hab, 0 );

WinRegisterClass( hab, szAppName, ClientWndProc,
                  CS_CLIPSIBLINGS | CS_CLIPCHILDREN | CS_SIZEREDRAW, 0 );

hwndFrame = WinCreateStdWindow( HWND_DESKTOP, WS_VISIBLE, &FrameFlags,
                                szAppName, szAppName, 0L,
                                NULLHANDLE, ID_PLATFORM, &hwndClient );

WinLockupSystem( hab );      /* Don't allow user to interact with
                               application unless password is known */

/* Application code here ... */

WinUnlockSystem( hab, pszPassword ); /* Unlock system when ready */
.
.
.
```

WinLockupSystem - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinLockVisRegions

WinLockVisRegions - Syntax

This function locks or unlocks the visible regions of all the windows on the screen, preventing any of the visible regions from changing.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndDesktop; /* Desktop-window handle or HWND_DESKTOP. */
BOOL    fLock;        /* Indicates whether the visible regions are being locked or unlocked. */
BOOL    rc;           /* Success indicator. */

rc = WinLockVisRegions(hwndDesktop, fLock);
```

WinLockVisRegions Parameter - hwndDesktop

hwndDesktop ([HWND](#)) - input
Desktop-window handle or HWND_DESKTOP.

WinLockVisRegions Parameter - fLock

fLock ([BOOL](#)) - input
Indicates whether the visible regions are being locked or unlocked.

TRUE	Lock the visible regions
FALSE	Unlock the visible regions.

WinLockVisRegions Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful.
FALSE	An error occurred.

WinLockVisRegions - Parameters

hwndDesktop ([HWND](#)) - input
Desktop-window handle or `HWND_DESKTOP`.

fLock ([BOOL](#)) - input
Indicates whether the visible regions are being locked or unlocked.

TRUE	Lock the visible regions
FALSE	Unlock the visible regions.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful.
FALSE	An error occurred.

WinLockVisRegions - Remarks

This function is useful to threads that need to prevent window visible regions from changing while some screen operation, such as copying screen pels into a memory bit map, is being performed.

Any other thread that tries to alter the visible regions is blocked while the visible regions are locked. While the visible regions are locked, no messages must be sent and no functions called that can send messages.

Only one thread can lock the visible regions at any one time. The same thread can call `WinLockVisRegions` multiple times. A lock count is maintained by the system and is incremented each time a locking call is made, and decremented each time an unlocking call is made. The visible regions are unlocked when the count is zero.

Note: Locking the visible regions does not prevent painting of a window by another process.

WinLockVisRegions - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinLockVisRegions - Related Functions

Related Functions

- [WinBeginPaint](#)
 - [WinEnableWindowUpdate](#)
 - [WinEndPaint](#)
 - [WinExcludeUpdateRegion](#)
 - [WinGetClipPS](#)
 - [WinGetPS](#)
 - [WinGetScreenPS](#)
 - [WinInvalidateRect](#)
 - [WinInvalidateRegion](#)
 - [WinIsWindowShowing](#)
 - [WinIsWindowVisible](#)
 - [WinLockVisRegions](#)
 - [WinOpenWindowDC](#)
 - [WinQueryUpdateRect](#)
 - [WinQueryUpdateRegion](#)
 - [WinRealizePalette](#)
 - [WinReleasePS](#)
 - [WinShowWindow](#)
 - [WinUpdateWindow](#)
 - [WinValidateRect](#)
 - [WinValidateRegion](#)
-

WinLockVisRegions - Example Code

This example uses WinLockVisRegions to prevent any window's visible region from changing while a screen operation is executing. WinLockVisRegions is called before the screen operation to lock the visible regions and again after the operation to unlock the regions.

```
#define INCL_WINWINDOWMGR          /* Window Manager Functions */
#include <os2.h>

BOOL fSuccess;                    /* success indicator */

/* lock visible regions */
fSuccess = WinLockVisRegions(HWND_DESKTOP, TRUE);

/*
 *
 * . executing screen operation
 *
 */

/* unlock visible regions */
fSuccess = WinLockVisRegions(HWND_DESKTOP, FALSE);
```

WinLockVisRegions - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinLockWindowUpdate

WinLockWindowUpdate - Syntax

This function disables or enables output to a window and its descendants.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndDeskTop;      /* Desktop handle of the screen containing the window to be locked. */
HWND    hwndLockUpdate;   /* Handle of window in which output is to be prevented. */
BOOL    rc;               /* Success indicator. */

rc = WinLockWindowUpdate(hwndDeskTop, hwndLockUpdate);
```

WinLockWindowUpdate Parameter - hwndDeskTop

hwndDeskTop ([HWND](#)) - input
Desktop handle of the screen containing the window to be locked.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

WinLockWindowUpdate Parameter - hwndLockUpdate

hwndLockUpdate ([HWND](#)) - input
Handle of window in which output is to be prevented.

NULLHANDLE	Enable output in the locked window and its descendants.
Other	Handle of the window in which output is to be prevented. Output is also prevented in the descendants of the

window.

WinLockWindowUpdate Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful operation.
FALSE	Error occurred.

WinLockWindowUpdate - Parameters

hwndDeskTop (**HWND**) - input
Desktop handle of the screen containing the window to be locked.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

hwndLockUpdate (**HWND**) - input
Handle of window in which output is to be prevented.

NULLHANDLE	Enable output in the locked window and its descendants.
Other	Handle of the window in which output is to be prevented. Output is also prevented in the descendants of the window.

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful operation.
FALSE	Error occurred.

WinLockWindowUpdate - Remarks

This function is used by threads that need to draw on an area of the screen over which they have no control. For example, the user interface sizing and moving calls use this call when drawing the shadow box, as a window is sized or moved.

All threads continue to run while the window is disabled; only output is prevented.

If one thread disables the window, other threads using this function are blocked until the first enables the window, although they can still receive messages.

This function does not prevent screen group switches, because these may be necessary to handle "hard errors" in other screen groups.

WinLockWindowUpdate - Example Code

This example disables output to a window and its children during a move operation (WM_MOVE) and then re-enables output once the move is finished.

```
#define INCL_WINWINDOWMGR          /* Window Manager Functions */
#include <os2.h>

HWND  hwndLock;          /* handle of window to be (un)locked */
BOOL  fSuccess;          /* success indicator */

case WM_MOVE:
    /* lock output */
    fSuccess = WinLockWindowUpdate(HWND_DESKTOP, hwndLock);

    /*
     *
     * . execute window move
     *
     */

    /* unlock output */
    fSuccess = WinLockWindowUpdate(HWND_DESKTOP, NULLHANDLE);
```

WinLockWindowUpdate - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

WinMakePoints

WinMakePoints - Syntax

This function converts points to graphics points.

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;    /* Anchor-block handle. */
PPOINTL  pwpt;   /* Points to be converted. */
ULONG    cwpt;   /* Number of points to be converted. */
BOOL      rc;    /* Success indicator. */

rc = WinMakePoints(hab, pwpt, cwpt);
```

WinMakePoints Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinMakePoints Parameter - pwpt

pwpt ([PPOINTL](#)) - in/out
Points to be converted.

The data type of these points after conversion is [POINTL](#).

WinMakePoints Parameter - cwpt

cwpt ([ULONG](#)) - input
Number of points to be converted.

Must be positive.

WinMakePoints Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinMakePoints - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

pwpt ([PPOINTL](#)) - in/out
Points to be converted.

The data type of these points after conversion is [POINTL](#).

cwpt ([ULONG](#)) - input
Number of points to be converted.

Must be positive.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinMakePoints - Remarks

This function converts the array of points from a [WPOINT](#) data structure into a [POINTL](#) data structure.

WinMakePoints - Example Code

This example calls WinMakePoints to convert a 3-element array of points from window points (WPOINT structure) to graphics points (POINTL structure).

```
#define INCL_WINRECTANGLES      /* Window Rectangle Functions */
#include <os2.h>

HAB      hab;                  /* anchor-block handle */
BOOL  fSuccess;                /* success indicator */
/* array of window points */
WPOINT pwptppt[3] = {0,0,0,0,20,0,50,0,100,0,60,0};

/* convert points */
fSuccess = WinMakePoints(hab, pwptppt, 3);
```

WinMakePoints - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

WinMakeRect

WinMakeRect - Syntax

This function converts a rectangle to a graphics rectangle.

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;    /* Anchor-block handle. */
PRECTL   pwrc;   /* Rectangle to be converted. */
BOOL     rc;     /* Success indicator. */

rc = WinMakeRect(hab, pwrc);
```

WinMakeRect Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinMakeRect Parameter - pwrc

pwrc ([PRECTL](#)) - in/out
Rectangle to be converted.

The data type of the rectangle after conversion is [RECTL](#).

WinMakeRect Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinMakeRect - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

pwr ([RECTL](#)) - in/out
Rectangle to be converted.

The data type of the rectangle after conversion is [RECTL](#).

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinMakeRect - Remarks

This function converts a rectangle from a [WRECT](#) data structure into a [RECTL](#) data structure.

WinMakeRect - Related Functions

Related Functions

- [WinCopyRect](#)
 - [WinEqualRect](#)
 - [WinFillRect](#)
 - [WinInflateRect](#)
 - [WinIntersectRect](#)
 - [WinIsRectEmpty](#)
 - [WinOffsetRect](#)
 - [WinPtInRect](#)
 - [WinSetRect](#)
 - [WinSetRectEmpty](#)
 - [WinSubtractRect](#)
 - [WinUnionRect](#)
-

WinMakeRect - Example Code

This example calls WinMakeRect to convert a window rectangle (WRECT structure) to a graphics rectangle (RECTL structure).

```
#define INCL_WINRECTANGLES      /* Window Rectangle Functions */
#include <os2.h>

HAB      hab;                /* anchor-block handle */
BOOL  fSuccess;              /* success indicator */
/* window rectangle */
WRECT  pwrpcrc = {0,0,0,0,50,0,50,0};

/* convert rectangle */
fSuccess = WinMakeRect(hab, &pwrpcrc);
```

WinMakeRect - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinMapDlgPoints

WinMapDlgPoints - Syntax

This function maps points from dialog coordinates to window coordinates, or from window coordinates to dialog coordinates.

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndDlg;          /* Dialog-window handle. */
PPOINTL   prgwptl;          /* Coordinate points to be mapped. */
ULONG      cwpt;            /* Number of coordinate points. */
BOOL      fCalcWindowCoords; /* Calculation control. */
BOOL      rc;               /* Coordinates-mapped indicator. */

rc = WinMapDlgPoints(hwndDlg, prgwptl, cwpt,
    fCalcWindowCoords);
```

WinMapDlgPoints Parameter - hwndDlg

hwndDlg ([HWND](#)) - input
Dialog-window handle.

WinMapDlgPoints Parameter - prgwptl

prgwptl ([PPOINTL](#)) - in/out
Coordinate points to be mapped.

The mapped points are substituted.

WinMapDlgPoints Parameter - cwpt

cwpt (**ULONG**) - input
Number of coordinate points.

WinMapDlgPoints Parameter - fCalcWindowCoords

fCalcWindowCoords (**BOOL**) - input
Calculation control.

TRUE

The points are in dialog coordinates and are to be mapped into window coordinates relative to the window specified by the *hwndDlg* parameter.

FALSE

The points are in window coordinates relative to the window specified by the *hwndDlg* parameter and are to be mapped into dialog coordinates.

WinMapDlgPoints Return Value - rc

rc (**BOOL**) - returns
Coordinates-mapped indicator.

TRUE

Coordinates successfully mapped

FALSE

Coordinates not successfully mapped.

WinMapDlgPoints - Parameters

hwndDlg (**HWND**) - input
Dialog-window handle.

prgwpptl (**PPOINTL**) - in/out
Coordinate points to be mapped.

The mapped points are substituted.

cwpt (**ULONG**) - input
Number of coordinate points.

fCalcWindowCoords ([BOOL](#)) - input
Calculation control.

TRUE	The points are in dialog coordinates and are to be mapped into window coordinates relative to the window specified by the <i>hwndDlg</i> parameter.
FALSE	The points are in window coordinates relative to the window specified by the <i>hwndDlg</i> parameter and are to be mapped into dialog coordinates.

rc ([BOOL](#)) - returns
Coordinates-mapped indicator.

TRUE	Coordinates successfully mapped
FALSE	Coordinates not successfully mapped.

WinMapDlgPoints - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinMapDlgPoints - Related Functions

Related Functions

- [WinMapDlgPoints](#)
- [WinMapWindowPoints](#)

WinMapDlgPoints - Example Code

This example calls WinMapDlgPoints to map a point from dialog coordinates to window coordinates relative to the dialog window.

```
#define INCL_WINDIALOGS          /* Window Dialog Mgr Functions */
#include <os2.h>

HWND  hwndDlg;                  /* handle of dialog window */
BOOL  fSuccess;                 /* success indicator */
POINTL aptlPoint;              /* point to be mapped */

/* map point to relative window coordinates */
fSuccess = WinMapDlgPoints(hwndDlg, &aptlPoint, 1, TRUE);
```

WinMapDlgPoints - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinMapWindowPoints

WinMapWindowPoints - Syntax

This function maps a set of points from a coordinate space relative to one window into a coordinate space relative to another window.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndFrom; /* Handle of the window from whose coordinates points are to be mapped. */
HWND      hwndTo;   /* Handle of the window to whose coordinates points are to be mapped. */
PPOINTL   prgptl;   /* Points to be mapped to the new coordinate system. */
LONG      cwpt;     /* Number of points to be mapped. */
BOOL      rc;       /* Success indicator. */

rc = WinMapWindowPoints(hwndFrom, hwndTo,
                        prgptl, cwpt);
```

WinMapWindowPoints Parameter - hwndFrom

hwndFrom ([HWND](#)) - input

Handle of the window from whose coordinates points are to be mapped.

HWND_DESKTOP

Points are mapped from screen coordinates

Other

Points are mapped from window coordinates.

WinMapWindowPoints Parameter - hwndTo

hwndTo ([HWND](#)) - input

Handle of the window to whose coordinates points are to be mapped.

HWND_DESKTOP	Points are mapped into screen coordinates
Other	Points are mapped into window coordinates.

WinMapWindowPoints Parameter - prgptl

prgptl ([PPOINTL](#)) - in/out
Points to be mapped to the new coordinate system.

WinMapWindowPoints Parameter - cwpt

cwpt ([LONG](#)) - input
Number of points to be mapped.

prgptl can be a [RECTL](#) structure, in which case this parameter should have the value 2.

Note: This is not supported in all languages.

WinMapWindowPoints Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE Successful completion
FALSE Error occurred.

WinMapWindowPoints - Parameters

hwndFrom ([HWND](#)) - input
Handle of the window from whose coordinates points are to be mapped.

HWND_DESKTOP Points are mapped from screen coordinates
Other Points are mapped from window coordinates.

hwndTo ([HWND](#)) - input
Handle of the window to whose coordinates points are to be mapped.

HWND_DESKTOP Points are mapped into screen coordinates

Other

Points are mapped into window coordinates.

prgptl ([PPOINTL](#)) - in/out

Points to be mapped to the new coordinate system.

cwpt ([LONG](#)) - input

Number of points to be mapped.

prgptl can be a [RECTL](#) structure, in which case this parameter should have the value 2.

Note: This is not supported in all languages.

rc ([BOOL](#)) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinMapWindowPoints - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)

An invalid window handle was specified.

WinMapWindowPoints - Related Functions

Related Functions

- [WinMapDlgPoints](#)
- [WinMapWindowPoints](#)

WinMapWindowPoints - Example Code

This example calls `WinMapWindowPoints` to map a mouse point on the desktop window to a mouse point in the client window and then checks whether the mouse pointer is inside the client area or not.

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#define INCL_WINRECTANGLES    /* Window Rectangle Functions */
#define INCL_WINPOINTERS      /* Window Pointer Functions */
#include <os2.h>

HAB      hab;           /* anchor-block handle */
HWND     hwndClient;    /* handle of client window */
BOOL     fSuccess;      /* success indicator */
POINTL   ptlMouse;      /* mouse pointer position */
RECTL    rclWork;       /* client area */

/* get current mouse pointer position */
WinQueryPointerPos(HWND_DESKTOP, &ptlMouse);

/* map from desktop to client window */
```

```
fSuccess = WinMapWindowPoints(HWND_DESKTOP, hwndClient,
                              &ptlMouse, 1);

/* check if new mouse position is inside the client area */
WinQueryWindowRect(hwndClient, &rclWork);
if (WinPtInRect(hab, &rclWork, &ptlMouse))
{
    /* pointer is in client area */
}
```

WinMapWindowPoints - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinMessageBox

WinMessageBox - Syntax

This function creates, displays, and operates a message box window.

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND    hwndParent; /* Parent-window handle of the created message-box window. */
HWND    hwndOwner;  /* Requested owner-window handle of the created message-box window. */
PSZ     pszText;     /* Message-box window message. */
PSZ     pszCaption;  /* Message-box window title. */
ULONG   idWindow;    /* Message-box window identity. */
ULONG   flStyle;     /* Message-box window style. */
ULONG   usResponse;  /* User-response value. */

usResponse = WinMessageBox(hwndParent, hwndOwner,
                          pszText, pszCaption, idWindow,
                          flStyle);
```

WinMessageBox Parameter - hwndParent

hwndParent ([HWND](#)) - input
Parent-window handle of the created message-box window.

[HWND_DESKTOP](#)
The message box is to be main window.

Other
Parent-window handle.

WinMessageBox Parameter - hwndOwner

hwndOwner ([HWND](#)) - input
Requested owner-window handle of the created message-box window.

The actual owner window is calculated using the algorithm specified in the description of the [WinLoadDlg](#) function.

WinMessageBox Parameter - pszText

pszText ([PSZ](#)) - input
Message-box window message.

The text of the message to be displayed within the message-box window. If multiple lines are required, carriage-return characters must be inserted into the text at appropriate points.

WinMessageBox Parameter - pszCaption

pszCaption ([PSZ](#)) - input
Message-box window title.

The text for the title should not be longer than 40 characters. If text longer than this is supplied, text centering is still performed, even though the beginning and end of the string are not visible.

NULL
The text Error is to be displayed as the title of the message-box window.

Other
The text to be displayed as the title of the message-box window.

WinMessageBox Parameter - idWindow

idWindow ([ULONG](#)) - input
Message-box window identity.

This value is passed to the HK_HELP hook if the [WM_HELP](#) message is received by the message-box window. It must be greater or equal to 0 and less or equal to 0xFFFF.

WinMessageBox Parameter - flStyle

flStyle ([ULONG](#)) - input
Message-box window style.

These values may be combined using the logical-OR operation but only one value can be taken from each of the following groups.

Button or Button Group

MB_OK	Message box contains an OK push button.
MB_OKCANCEL	Message box contains both OK and CANCEL push buttons.
MB_CANCEL	Message box contains a CANCEL push button.
MB_ENTER	Message box contains an ENTER push button.
MB_ENTERCANCEL	Message box contains both ENTER and CANCEL push buttons.
MB_RETRYCANCEL	Message box contains both RETRY and CANCEL push buttons.
MB_ABORTRETRYIGNORE	Message box contains ABORT, RETRY, and IGNORE push buttons.
MB_YESNO	Message box contains both YES and NO push buttons.
MB_YESNOCANCEL	Message box contains YES, NO, and CANCEL push buttons.

Help button

MB_HELP	Message box contains a HELP push button. When this is selected a WM_HELP message is sent to the window procedure of the message box.
---------	--

Color or Icon

MB_ERROR	Message box contains a small red circle with a red line across it.
MB_ICONASTERISK	Message box contains an information (i) icon.
MB_ICONEXCLAMATION	Message box contains an exclamation point (!) icon.
MB_ICONHAND	Message box contains a small red circle with a red line across it.
MB_ICONQUESTION	Message box contains a question mark (?) icon.
MB_INFORMATION	Message box contains an information (i) icon.
MB_NOICON	Message box is not to contain an icon.
MB_QUERY	Message box contains a question mark (?) icon.
MB_WARNING	Message box contains an exclamation point (!) icon.

Default action

MB_DEFBUTTON1	The first button is the default selection. This is the default case, if none of MB_DEFBUTTON1, MB_DEFBUTTON2, and MB_DEFBUTTON3 is specified.
MB_DEFBUTTON2	The second button is the default selection.
MB_DEFBUTTON3	The third button is the default selection.

Modality indicator

MB_APPLMODAL

Message box is application modal. This is the default case. Its owner is disabled; therefore, do not specify the owner as the parent if this option is used.

MB_SYSTEMMODAL

Message box is system modal.

Mobility indicator

MB_MOVEABLE

Message box is moveable.

The message box is displayed with a title bar and a system menu, which shows only the Move, Close, and Task Manager choices, which can be selected either by use of the pointing device or by accelerator keys.

If the user selects Close, the message box is removed and the *usResponse* is set to MBID_CANCEL, whether or not a cancel button existed within the message box.

WinMessageBox Return Value - usResponse

usResponse (**ULONG**) - returns
User-response value.

MBID_ENTER	ENTER push button was selected
MBID_OK	OK push button was selected
MBID_CANCEL	CANCEL push button was selected
MBID_ABORT	ABORT push button was selected
MBID_RETRY	RETRY push button was selected
MBID_IGNORE	IGNORE push button was selected
MBID_YES	YES push button was selected
MBID_NO	NO push button was selected
MBID_ERROR	Function not successful; an error occurred.

WinMessageBox - Parameters

hwndParent (**HWND**) - input
Parent-window handle of the created message-box window.

HWND_DESKTOP	The message box is to be main window.
Other	Parent-window handle.

hwndOwner (**HWND**) - input
Requested owner-window handle of the created message-box window.

The actual owner window is calculated using the algorithm specified in the description of the [WinLoadDlg](#) function.

pszText (PSZ) - input

Message-box window message.

The text of the message to be displayed within the message-box window. If multiple lines are required, carriage-return characters must be inserted into the text at appropriate points.

pszCaption (PSZ) - input

Message-box window title.

The text for the title should not be longer than 40 characters. If text longer than this is supplied, text centering is still performed, even though the beginning and end of the string are not visible.

NULL

The text Error is to be displayed as the title of the message-box window.

Other

The text to be displayed as the title of the message-box window.

idWindow (ULONG) - input

Message-box window identity.

This value is passed to the HK_HELP hook if the [WM_HELP](#) message is received by the message-box window. It must be greater or equal to 0 and less or equal to 0xFFFF.

flStyle (ULONG) - input

Message-box window style.

These values may be combined using the logical-OR operation but only one value can be taken from each of the following groups.

Button or Button Group

MB_OK	Message box contains an OK push button.
MB_OKCANCEL	Message box contains both OK and CANCEL push buttons.
MB_CANCEL	Message box contains a CANCEL push button.
MB_ENTER	Message box contains an ENTER push button.
MB_ENTERCANCEL	Message box contains both ENTER and CANCEL push buttons.
MB_RETRYCANCEL	Message box contains both RETRY and CANCEL push buttons.
MB_ABORTRETRYIGNORE	Message box contains ABORT, RETRY, and IGNORE push buttons.
MB_YESNO	Message box contains both YES and NO push buttons.
MB_YESNOCANCEL	Message box contains YES, NO, and CANCEL push buttons.

Help button

MB_HELP	Message box contains a HELP push button. When this is selected a WM_HELP message is sent to the window procedure of the message box.
---------	--

Color or Icon

MB_ERROR	Message box contains a small red circle with a red line across it.
MB_ICONASTERISK	Message box contains an information (i) icon.
MB_ICONEXCLAMATION	Message box contains an exclamation point (!) icon.
MB_ICONHAND	Message box contains a small red circle with a red line across it.
MB_ICONQUESTION	Message box contains a question mark (?) icon.
MB_INFORMATION	Message box contains an information (i) icon.

MB_NOICON	Message box is not to contain an icon.
MB_QUERY	Message box contains a question mark (?) icon.
MB_WARNING	Message box contains an exclamation point (!) icon.
Default action	
MB_DEFBUTTON1	The first button is the default selection. This is the default case, if none of MB_DEFBUTTON1, MB_DEFBUTTON2, and MB_DEFBUTTON3 is specified.
MB_DEFBUTTON2	The second button is the default selection.
MB_DEFBUTTON3	The third button is the default selection.
Modality indicator	
MB_APPLMODAL	Message box is application modal. This is the default case. Its owner is disabled; therefore, do not specify the owner as the parent if this option is used.
MB_SYSTEMMODAL	Message box is system modal.
Mobility indicator	
MB_MOVEABLE	<p>Message box is moveable.</p> <p>The message box is displayed with a title bar and a system menu, which shows only the Move, Close, and Task Manager choices, which can be selected either by use of the pointing device or by accelerator keys.</p> <p>If the user selects Close, the message box is removed and the <i>usResponse</i> is set to MBID_CANCEL, whether or not a cancel button existed within the message box.</p>
usResponse (ULONG) - returns User-response value.	
MBID_ENTER	ENTER push button was selected
MBID_OK	OK push button was selected
MBID_CANCEL	CANCEL push button was selected
MBID_ABORT	ABORT push button was selected
MBID_RETRY	RETRY push button was selected
MBID_IGNORE	IGNORE push button was selected
MBID_YES	YES push button was selected
MBID_NO	NO push button was selected
MBID_ERROR	Function not successful; an error occurred.

WinMessageBox - Remarks

The message box consists of a message and a simple dialog with the user.

This function behaves in a similar way to [WinDlgBox](#), and the remarks concerning modality which are documented under that call, and also under the [WinLoadDlg](#) and [WinProcessDlg](#) functions, also apply here.

This function should not be used while pointing device capture is set (see [WinSetCapture](#)).

If the keyboard is used to cycle from one window to the next, the message box and its parent window are considered to be next to each other in the sequence.

If a message box is created as part of the processing of a dialog window, where the dialog window has not been dismissed, the dialog window should be made the owner of the message box window.

If a system modal message box is created to indicate to the user that the system is running out of memory, the strings passed into this call must not be taken from a resource file, as an attempt to load the resource file could fail because of the lack of memory. However, such a message box can safely use the hand icon because this icon is always memory-resident.

The size of the message box is determined as follows:

- The minimum width of a message box is enough to display 40 characters of average width.
- The minimum height of a message box is enough to display 2 lines.
- The text of a message box is word-wrapped by default. If more than two lines are required to display the text, the height of the message box is increased up to a maximum of two thirds of the screen height. The height of a message box can never exceed this value.
- If necessary, the width of a message box is increased to allow room to display the title.

Text is wrapped at word boundaries (spaces). If a word is too big to fit on one line, the start of the word is not wrapped to the next line, but stays adjacent to the text it follows, and the word is split at the box boundary.

The message box is centered on the screen.

If a message box window has a CANCEL button, the MBID_CANCEL value is returned if either the Escape key is pressed or the Cancel button is selected. If the message box window has no CANCEL button, pressing the Escape key has no effect.

Note: If a dialog box or a message box is up for a window, and the parent or owner of that window is destroyed, the code following the WinDlgBox or WinMessageBox call is executed even though the parent/owner window no longer exists. This can result in accessing data that no longer exists; especially data referenced in the window words. Therefore, it is extremely important to determine the state of your child-window procedure after this function returns. The most straightforward method for doing this is to call WinQueryWindowPtr to get a pointer to the window words. If the returned pointer is NULL, then you should exit immediately. Should this be the case, the bottom-up rule (that is, the child window gets WM_DESTROY messages first, then the parent window) still applies, and it becomes the child window procedure's responsibility to exit gracefully.

WinMessageBox - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_INVALID_FLAG (0x1019)
An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

WinMessageBox - Related Functions

Related Functions

- [WinAlarm](#)
- [WinFlashWindow](#)
- [WinMessageBox](#)

WinMessageBox - Example Code

This example shows a typical use of the WinMessageBox function when debugging an application. The C run-time function sprintf is used to format the body of the message. In this case, it converts the coordinates of the mouse pointer (retrieved with the WinQueryPointerPos function) into a string. The string is then displayed by calling WinMessageBox.

```
#define INCL_WINDIALOGS          /* Window Dialog Mgr Functions */
#define INCL_WINPOINTERS        /* Window Pointer Functions */
#include <os2.h>

CHAR szMsg[100];                /* message */
POINTL ptl;                     /* message data */
HWND hwndClient;                /* client window handle */

WinQueryPointerPos(HWND_DESKTOP, &ptl);
sprintf(szMsg, "x = %ld y = %ld", ptl.x, ptl.y);
WinMessageBox(HWND_DESKTOP,
    hwndClient,                  /* client-window handle */
    szMsg,                       /* body of the message */
    "Debugging information",     /* title of the message */
    0,                           /* message box id */
    MB_NOICON | MB_OK);          /* icon and button flags */
```

WinMessageBox - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinMessageBox2

WinMessageBox2 - Syntax

This function creates a message-box window that can be used to display error messages and ask questions.

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_GPI, INCL_WINWINDOWMGR, */
#include <os2.h>

HWND hwndParent; /* Parent-window handle of the message-box window to be created. */
HWND hwndOwner; /* Requested owner-window handle of the message-box window to be created. */
PSZ pszText; /* Message-box window message. */
```

```

PSZ      pszTitle;      /* Message-box window title. */
ULONG    ulWindow;      /* Message-box window identity. */
PMB2INFO pmb2info;      /* Input structure for message-box window. */
ULONG    ulButtonId;    /* Id of the button that was clicked, or MBID_ERROR. */

ulButtonId = WinMessageBox2(hwndParent, hwndOwner,
    pszText, pszTitle, ulWindow,
    pmb2info);

```

WinMessageBox2 Parameter - hwndParent

hwndParent ([HWND](#)) - input
Parent-window handle of the message-box window to be created.

HWND-DESKTOP
The message box is to be main window.

Other
Parent-window handle.

WinMessageBox2 Parameter - hwndOwner

hwndOwner ([HWND](#)) - input
Requested owner-window handle of the message-box window to be created.

The actual owner window is calculated using the algorithm specified in the description of the [WinLoadDlg](#) function.

WinMessageBox2 Parameter - pszText

pszText ([PSZ](#)) - input
Message-box window message.

The text of the message to be displayed within the message-box window. If multiple lines are required, carriage-return characters must be inserted into the text at appropriate points.

WinMessageBox2 Parameter - pszTitle

pszTitle ([PSZ](#)) - input
Message-box window title.

The text for the title should not be longer than 40 characters. If text longer than this is supplied, text centering is still performed, even though the beginning and end of the string are not visible.

A value if NULL indicates that the text "Error" is to be displayed as the title of the message-box window.

WinMessageBox2 Parameter - ulWindow

ulWindow ([ULONG](#)) - input
Message-box window identity.

This value is passed to the HK_HELP hook if the [WM_HELP](#) message is received by the message-box window.

WinMessageBox2 Parameter - pmb2info

pmb2info ([PMB2INFO](#)) - input
Input structure for message-box window.

WinMessageBox2 Return Value - ulButtonId

ulButtonId ([ULONG](#)) - returns
Id of the button that was clicked, or MBID_ERROR.

WinMessageBox2 - Parameters

hwndParent ([HWND](#)) - input
Parent-window handle of the message-box window to be created.

HWND-DESKTOP
The message box is to be main window.
Other
Parent-window handle.

hwndOwner ([HWND](#)) - input
Requested owner-window handle of the message-box window to be created.

The actual owner window is calculated using the algorithm specified in the description of the [WinLoadDlg](#) function.

pszText ([PSZ](#)) - input
Message-box window message.

The text of the message to be displayed within the message-box window. If multiple lines are required, carriage-return characters must be inserted into the text at appropriate points.

pszTitle ([PSZ](#)) - input
Message-box window title.

The text for the title should not be longer than 40 characters. If text longer than this is supplied, text centering is still performed, even though the beginning and end of the string are not visible.

ulWindow (**ULONG**) - input
Message-box window identity.

pmb2info (PMB2INFO) - input
Input structure for mesage-box window.

=====

=====

```
#define INCL_WINDIALOGS /* Window Dialog Mgr functions*/
#define INCL_WINPOINTERS /* Window Pointer functions */
#define NUM_BTN 4
#include <os2.h>

ULONG ulResult; /* Indicates which button to */
/* push on the message box */
```

```

ULONG    i;                                /* A loop index */
MB2INFO *pmbInfo;                          /* Pointer to the message box structure */

PSZ       pszBoxTitle    = "A title with up to 40 characters in it ";

MB2D mb2dBut[NUM_BUT] =
{
    /* Static copy of button definitions */
    { "AAAA", ID_BUTTON1, BS_PUSHTYPE}, /* Or use 0 */
    { "BBBBBBBBBBBBBBBB", ID_BUTTON2, BS_DEFAULT},
    { "CCCCCCC", ID_BUTTON3, 0},
    { "D", ID_BUTTON4, 0}
};

/* Size of the message box structure needed. */
/* Need space for MB2INFO plus 3 additional buttons */
/* (one button is defined in the MB2INFO structure). */
ULONG ulInfoSize = (sizeof(MB2INFO) + (sizeof(MB2D) * (NUM_BUT-1)));

/* Allocate space for the MB2INFO structure */
pmbInfo = malloc (ulInfoSize);

pmbInfo->cb = ulInfoSize; /* Size of block */
pmbInfo->hIcon = WinLoadPointer(HWND_DESKTOP, 0, ID_ICON1);
pmbInfo->cButtons = NUM_BUT; /* Number of buttons for box */
pmbInfo->flStyle = MB_CUSTOMICON |
                MB_MOVEABLE;
pmbInfo->hwndNotify = NULLHANDLE;

/* Copy information for each button to the MB2INFO structure */
for (i = 0; i < NUM_BUT; i++)
{
    memcpy( pmbInfo->mb2d+i , mb2dBut+i , sizeof(MB2D));
};

ulResult = WinMessageBox2(HWND_DESKTOP, /* Parent window */
                          HWND_DESKTOP, /* Owner window */
                          (PSZ)"Line 1 of your message.\nLine 2 of message.",
                          pszBoxTitle, /* Message box title */
                          1234L, /* Identifier for message box */
                          pmbInfo); /* Button definitions for message box */

```

WinMessageBox2 - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinMoveObject

WinMoveObject - Syntax

This function is specific to OS/2 Version 3.0 or higher.

This function moves an object from its existing location to a specified new destination.

```
#define INCL_WINWORKPLACE
#include <os2.h>

HOBJECT    hObjectofObject; /* Handle of the Workplace Shell object being moved. */
HOBJECT    hObjectofDest;  /* Handle of the destination folder into which hObjectofObject is to be moved. */
ULONG      ulReserved;     /* Reserved value; must be 0. */
HOBJECT    rc;             /* Handle of the source object being moved. */

rc = WinMoveObject(hObjectofObject, hObjectofDest,
    ulReserved);
```

WinMoveObject Parameter - hObjectofObject

hObjectofObject (**HOBJECT**) - input
Handle of the Workplace Shell object being moved.

WinMoveObject Parameter - hObjectofDest

hObjectofDest (**HOBJECT**) - input
Handle of the destination folder into which *hObjectofObject* is to be moved.

WinMoveObject Parameter - ulReserved

ulReserved (**ULONG**) - input
Reserved value; must be 0.

WinMoveObject Return Value - rc

rc (**HOBJECT**) - returns
Handle of the source object being moved.

A return value of NULLHANDLE indicates that either *hObjectofDest* is NULLHANDLE or an object with the same name as *hObjectofObject* exists in the destination folder.

WinMoveObject - Parameters

hObjectofObject ([HOBJECT](#)) - input

Handle of the Workplace Shell object being moved.

hObjectofDest ([HOBJECT](#)) - input

Handle of the destination folder into which *hObjectofObject* is to be moved.

ulReserved ([ULONG](#)) - input

Reserved value; must be 0.

rc ([HOBJECT](#)) - returns

Handle of the source object being moved.

A return value of NULLHANDLE indicates that either *hObjectofDest* is NULLHANDLE or an object with the same name as *hObjectofObject* exists in the destination folder.

WinMoveObject - Remarks

Using [HOBJECT](#) for .INI files or files in which an application uses a rename/save/delete sequence is not supported. Its REXX counterpart is SysMoveObject.

WinMoveObject - Errors

Possible returns from [WinGetLastError](#)

WPERR_INVALID_FLAGS (0x1719)

An invalid flag was specified.

WinMoveObject - Related Functions

Related Functions

- [WinCopyObject](#)
- [WinCreateObject](#)
- [WinDestroyObject](#)
- [WinQueryObjectWindow](#)
- [WinSaveObject](#)

WinMoveObject - Example Code

This example moves the drives object onto the Desktop.

```
#define INCL_WINWORKPLACE
#include "os2.h"

HOBJECT    hObjectofDest;
HOBJECT    hObjectofObject;
HOBJECT    hObjectofResult;

hObjectofObject = WinQueryObject("<WP_DRIVES>");
if (hObjectofObject != NULL)
{
    /* WinQueryObject of Drives was successful */
    hObjectofDest = WinQueryObject("<WP_DESKTOP>");

    if (hObjectofDest != NULL)
    {
        /* WinQueryObject of Startup was successful */
        hObjectofResult = WinMoveObject(hObjectofObject, hObjectofDest, 0);

        if (hObjectofResult != NULL)
        {
            /* Drives Object was successfully copied to the Desktop */
        }
        else
        {
            /* Move failed */
        }
    }
}
```

WinMoveObject - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinMultWindowFromIDs

WinMultWindowFromIDs - Syntax

This function finds the handles of child windows that belong to a specified window and have window identities within a specified range.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
```

```
HWND    hwndParent; /* Parent-window handle. */
HWND    prghwnd; /* Window handles. */
ULONG    idFirst; /* First window identity value in the range (inclusive). */
ULONG    idLast; /* Last window identity value in the range (inclusive). */
LONG     lWindows; /* Number of window handles returned. */

lWindows = WinMultWindowFromIDs(hwndParent,
                                prghwnd, idFirst, idLast);
```

WinMultWindowFromIDs Parameter - hwndParent

hwndParent (**HWND**) - input
Parent-window handle.

WinMultWindowFromIDs Parameter - prghwnd

prghwnd (**HWND**) - output
Window handles.

Array of window handles. The array must contain (*idLast* - *idFirst* + 1) elements. The handle of a window, whose identity is *WID* (in the range *idFirst* to *idLast*), has a zero-based index in the array of (*WID* - *idFirst*). If there is no window for a window identity within the range, the corresponding element in the array is `NULLHANDLE`.

WinMultWindowFromIDs Parameter - idFirst

idFirst (**ULONG**) - input
First window identity value in the range (inclusive).

It must be greater or equal to 0 and less or equal to 0xFFFF.

WinMultWindowFromIDs Parameter - idLast

idLast (**ULONG**) - input
Last window identity value in the range (inclusive).

It must be greater or equal to 0 and less or equal to 0xFFFF.

WinMultWindowFromIDs Return Value - lWindows

IWindows ([LONG](#)) - returns
Number of window handles returned.

0	No window handles returned
Other	Number of window handles returned.

WinMultWindowFromIDs - Parameters

hwndParent ([HWND](#)) - input
Parent-window handle.

prghwnd ([PHWND](#)) - output
Window handles.

Array of window handles. The array must contain (*idLast* - *idFirst* + 1) elements. The handle of a window, whose identity is WID (in the range *idFirst* to *idLast*), has a zero-based index in the array of (WID - *idFirst*). If there is no window for a window identity within the range, the corresponding element in the array is NULLHANDLE.

idFirst ([ULONG](#)) - input
First window identity value in the range (inclusive).

It must be greater or equal to 0 and less or equal to 0xFFFF.

idLast ([ULONG](#)) - input
Last window identity value in the range (inclusive).

It must be greater or equal to 0 and less or equal to 0xFFFF.

IWindows ([LONG](#)) - returns
Number of window handles returned.

0	No window handles returned
Other	Number of window handles returned.

WinMultWindowFromIDs - Remarks

This function can be used to enumerate all the items in a dialog group, or to enumerate all the frame controls of a standard window. This function is faster than individual calls to the [WinWindowFromID](#) function.

WinMultWindowFromIDs - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinMultWindowFromIDs - Related Functions

Related Functions

- [WinBeginEnumWindows](#)
 - [WinEndEnumWindows](#)
 - [WinEnumDlgItem](#)
 - [WinGetNextWindow](#)
 - [WinIsChild](#)
 - [WinMultWindowFromIDs](#)
 - [WinQueryWindow](#)
 - [WinSetOwner](#)
 - [WinSetParent](#)
-

WinMultWindowFromIDs - Example Code

This example finds the handles of all frame controls of a specified window via the WinMultWindowFromIDs call. The handles are returned in an array of window handles, and after the call completes, the handle for the minmax control window is assigned to a variable if a handle for it was found (i.e. handle not equal to NULLHANDLE).

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#define INCL_WINFRAMEMGR      /* Window Frame Functions */
#include <os2.h>

HWND  hwndParent;           /* parent window */
HWND  ahwnd[FID_CLIENT-FID_SYSMENU]; /* window handle array */
HWND  hwndMinMax;          /* minmax control window handle */
LONG  lHandles;             /* number of handles returned */

/* get all control handles between and including system menu and
   client windows */
lHandles = WinMultWindowFromIDs(hwndParent, ahwnd, FID_SYSMENU,
                                FID_CLIENT);

/* if any handles returned and the handle for the minmax control is
   not null, assign a variable to the minmax handle */
if (lHandles > 0 && ahwnd[FID_MINMAX - FID_SYSMENU] != NULLHANDLE)
    hwndMinMax = ahwnd[FID_MINMAX - FID_SYSMENU];
```

WinMultWindowFromIDs - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinNextChar

WinNextChar - Syntax

This function moves to the next character in a string.

```
#define INCL_WINCOUNTRY /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
ULONG    ulCodepage;    /* Code page. */
ULONG    ulCountry;     /* Reserved value, must be 0. */
PSZ      pszCurrentChar; /* Current character in a null-terminated string. */
PSZ      pszNextChar;   /* Pointer to the next character in the null-terminated string. */

pszNextChar = WinNextChar(hab, ulCodepage,
                          ulCountry, pszCurrentChar);
```

WinNextChar Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinNextChar Parameter - ulCodepage

ulCodepage (**ULONG**) - input
Code page.

If a non-existent code page is specified for *ulCodepage*, the code page specified by the queue associated with the calling thread is used.

WinNextChar Parameter - ulCountry

ulCountry (**ULONG**) - input
Reserved value, must be 0.

WinNextChar Parameter - pszCurrentChar

pszCurrentChar ([PSZ](#)) - input
Current character in a null-terminated string.

WinNextChar Return Value - pszNextChar

pszNextChar ([PSZ](#)) - returns
Pointer to the next character in the null-terminated string.

When the end of the string is reached, a pointer to the null-terminating character is returned.

WinNextChar - Parameters

- hab** ([HAB](#)) - input
Anchor-block handle.
- ulCodepage** ([ULONG](#)) - input
Code page.

If a non-existent code page is specified for *ulCodepage*, the code page specified by the queue associated with the calling thread is used.
- ulCountry** ([ULONG](#)) - input
Reserved value, must be 0.
- pszCurrentChar** ([PSZ](#)) - input
Current character in a null-terminated string.
- pszNextChar** ([PSZ](#)) - returns
Pointer to the next character in the null-terminated string.

When the end of the string is reached, a pointer to the null-terminating character is returned.

WinNextChar - Remarks

This function handles DBCS strings.

The following is the list of valid code pages:

Country	Code Page
Canadian-French	863
Desktop Publishing	1004
Iceland	861
Latin 1 Multilingual	850
Latin 2 Multilingual	852
Nordic	865

Portuguese	860
Turkey	857
U.S. (IBM PC)	437

Code page 1004 is compatible with Microsoft** Windows**.

The following EBCDIC code pages, based on character set 697, are also available for output:

Country	Code Page
Austrian/German	273
Belgian	500
Brazil	037
Czechoslovakia	870
Danish/Norwegian	277
Finnish/Swedish	278
French	297
Hungary	870
Iceland	871
International	500
Italian	280
Poland	870
Portuguese	037
Spanish	284
Turkey	1026
U.K.-English	285
U.S.-English	037
Yugoslavia	870

Code pages 274 (Belgian) and 282 (Portuguese) can be used to provide access to old data. The following is the list of valid country codes:

Country	Code
Arabic	785
Australian	61
Belgian	32
Canadian-French	2
Danish	45
Finnish	358
French	33
German	49
Hebrew	972
Italian	39
Japanese	81
Korean	82
Latin-American	3
Netherlands	31
Norwegian	47
Portuguese	351
Simpl. Chinese	86
Spanish	34
Swedish	46
Swiss	41
Trad. Chinese	88
UK-English	44
US-English	1
Other country	0

WinNextChar - Errors

Possible returns from [WinGetLastError](#)

PMERR_INV_CODEPAGE (0x2052)

An invalid code-page was specified.

This error can be obtained if using double-byte character set only.

PMERR_INVALID_STRING_PARM (0x100B)

The specified string parameter is invalid.

WinNextChar - Related Functions

Related Functions

- [WinCompareStrings](#)
 - [WinLoadString](#)
 - [WinNextChar](#)
 - [WinPrevChar](#)
 - [WinSubstituteStrings](#)
 - [WinUpper](#)
 - [WinUpperChar](#)
-

WinNextChar - Example Code

This example uses WinNextChar to traverse a string until a specified character is found, while maintaining an index to point to the character's position.

```
#define INCL_WINCOUNTRY /* Window Country Functions */
#include <os2.h>

HAB    hab; /* Anchor-block handle */
ULONG  idCodepage=437; /* Code page identity of both strings */
ULONG  idCountryCode=1; /* Country code */
char    pszString1[10]; /* First string */
char    *pszNextChar; /* Next character */
char    *pszCurrentChar; /* Current character */
ULONG  ulIndex; /* Array index */

/* Set initial values */
strcpy(pszString1, "Compare");
pszCurrentChar = pszString1;

ulIndex = 0;
do
{
    pszNextChar = WinNextChar(habMain, 437, 0, pszCurrentChar);

    /* The next line is necessary to get the next char */
    pszCurrentChar = pszNextChar;

    /* Set emergency escape route, just in case */
    ulIndex = ulIndex + 1;
    if (ulIndex >10)
        break;
} while (*pszNextChar != (char)NULL);
```

WinNextChar - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinOffsetRect

WinOffsetRect - Syntax

This function offsets a rectangle.

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */  
#include <os2.h>  
  
HAB      hab; /* Anchor-block handle. */  
PRECTL   prcl; /* Rectangle to be offset. */  
LONG     cx; /* x-value of offset. */  
LONG     cy; /* y-value of offset. */  
BOOL     rc; /* Success indicator. */  
  
rc = WinOffsetRect(hab, prcl, cx, cy);
```

WinOffsetRect Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinOffsetRect Parameter - prcl

prcl ([PRECTL](#)) - in/out
Rectangle to be offset.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) may also be used, if supported by the language.

WinOffsetRect Parameter - cx

cx (**LONG**) - input
x-value of offset.

WinOffsetRect Parameter - cy

cy (**LONG**) - input
y-value of offset.

WinOffsetRect Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinOffsetRect - Parameters

hab (**HAB**) - input
Anchor-block handle.

prcl (**PRECTL**) - in/out
Rectangle to be offset.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type **WRECT** may also be used, if supported by the language.

cx (**LONG**) - input
x-value of offset.

cy (**LONG**) - input
y-value of offset.

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinOffsetRect - Remarks

This function offsets the coordinates of *prc* by adding the value of the *cx* parameter to both the left and right coordinates, and the value of the *cy* to both the top and bottom coordinates.

WinOffsetRect - Related Functions

Related Functions

- [WinCopyRect](#)
- [WinEqualRect](#)
- [WinFillRect](#)
- [WinInflateRect](#)
- [WinIntersectRect](#)
- [WinIsRectEmpty](#)
- [WinOffsetRect](#)
- [WinPtInRect](#)
- [WinSetRect](#)
- [WinSetRectEmpty](#)
- [WinSubtractRect](#)
- [WinUnionRect](#)

WinOffsetRect - Example Code

This example moves a rectangle in response to the movement of the mouse (WM_MOUSEMOVE); the rectangle is moved (offset) based on the distance moved by the mouse since its previous position.

```
#define INCL_WINRECTANGLES      /* Window Rectangle Functions */
#include <os2.h>

int main(void)
{
    BOOL    fSuccess;           /* success indicator */
    HAB     hab;                /* anchor-block handle */
    RECTL   prclRect1 = {0,0,100,100}; /* rectangle */
    LONG    lcx;                /* Horizontal expansion */
    LONG    lcy;                /* Vertical expansion */
    POINTL   ptlPrev;           /* previous mouse position */
    POINTL   ptlCurr;           /* current mouse position */
    MPARAM   mpl;               /* Parameter 1 (x,y) point value */

    case WM_MOUSEMOVE:
        ptlCurr.x = (LONG) SHORT1FROMMP(mpl);
        ptlCurr.y = (LONG) SHORT2FROMMP(mpl);

        /* calculate distance from previous mouse position */
        lcx = (LONG)(ptlPrev.x - ptlCurr.x);
        lcy = (LONG)(ptlPrev.y - ptlCurr.y);

        fSuccess = WinOffsetRect(hab, &prclRect1, lcx, lcy);
```

WinOffsetRect - Topics

Select an item:

[Syntax](#)

[Parameters](#)

WinOpenClipbrd

WinOpenClipbrd - Syntax

This function opens the clipboard.

```
#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB    hab; /* Anchor-block handle. */
BOOL    rc; /* Success indicator. */

rc = WinOpenClipbrd(hab);
```

WinOpenClipbrd Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinOpenClipbrd Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Clipboard successfully opened
FALSE	Error occurred.

WinOpenClipbrd - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Clipboard successfully opened
FALSE	Error occurred.

WinOpenClipbrd - Remarks

The process reading the clipboard does not become the owner of the object in it; it must not update or free the object.

This function prevents other threads and processes from examining or changing the clipboard contents.

If another thread or process already has the clipboard open, this function does not return until the clipboard is closed.

Messages can be received from other threads and processes during the processing of this function.

WinOpenClipbrd - Related Functions

Related Functions

- [WinCloseClipbrd](#)
 - [WinEmptyClipbrd](#)
 - [WinEnumClipbrdFmts](#)
 - [WinOpenClipbrd](#)
 - [WinQueryClipbrdData](#)
 - [WinQueryClipbrdFmtInfo](#)
 - [WinQueryClipbrdOwner](#)
 - [WinQueryClipbrdViewer](#)
 - [WinSetClipbrdData](#)
 - [WinSetClipbrdOwner](#)
 - [WinSetClipbrdViewer](#)
-

WinOpenClipbrd - Example Code

This example opens the clipboard for use by the current process.

```
#define INCL_WINCLIPBOARD          /* Window Clipboard Functions */
#include <os2.h>

BOOL fSuccess;                    /* success indicator */
HAB hab;                          /* anchor-block handle */

fSuccess = WinOpenClipbrd(hab);
```

WinOpenClipbrd - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinOpenObject

WinOpenObject - Syntax

This function is specific to OS/2 Version 3.0 or higher.

This function either opens a view of the given object or surfaces an existing view.

```
#define INCL_WINWORKPLACE
#include <os2.h>

HOBJECT    hObjectofObject; /* Handle to a Workplace Shell object to be opened. */
ULONG      ulView;          /* View in which to open the specified object. */
BOOL       fFlags;          /* Flags. */
BOOL       rc;              /* Success indicator. */

rc = WinOpenObject(hObjectofObject, ulView,
                   fFlags);
```

WinOpenObject Parameter - hObjectofObject

hObjectofObject ([HOBJECT](#)) - input

Handle to a Workplace Shell object to be opened.

WinOpenObject Parameter - ulView

ulView ([ULONG](#)) - input

View in which to open the specified object.

OPEN_SETTINGS

OPEN_TREE

OPEN_DEFAULT
OPEN_CONTENTS
OPEN_DETAILS

WinOpenObject Parameter - fFlags

fFlags (BOOL) - input
Flags.

TRUE	Open a view of the object which already exists and supports concurrent views, by calling wpViewObject.
FALSE	Open a view of this object by calling wpOpen.

WinOpenObject Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinOpenObject - Parameters

hObjectofObject (HOBJECT) - input
Handle to a Workplace Shell object to be opened.

ulView (ULONG) - input
View in which to open the specified object.

OPEN_SETTINGS
OPEN_TREE
OPEN_DEFAULT
OPEN_CONTENTS
OPEN_DETAILS

fFlags (BOOL) - input
Flags.

TRUE

FALSE	Open a view of the object which already exists and supports concurrent views, by calling wpViewObject.
	Open a view of this object by calling wpOpen.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinOpenObject - Remarks

Using [HOBJECT](#) for .INI files or files in which an application uses a rename/save/delete sequence is not supported. Its REXX counterpart is SysOpenObject. If concurrent views is off, wpViewObject will resurface an existing object.

WinOpenObject - Related Functions

Related Functions

- [WinCreateObject](#)
 - [WinDestroyObject](#)
 - [WinSaveObject](#)
-

WinOpenObject - Example Code

```
#define INCL_WINWORKPLACE
#include "os2.h"

HOBJECT hObject;
ULONG    ulView;
BOOL     fFlags
BOOL     fSuccess;

hObject = WinQueryObject("<WP_DESKTOP>");
if (hObject != NULL)
{
    /* WinQueryObject was successful */

    fSuccess = WinOpenObject(hObject, OPEN_SETTINGS,TRUE);

    if (fSuccess)
    {
        /* If concurrent views is off and a settings view of the desktop already
        * exits, then it was successfully resurfaced.
        *
        * If concurrent views is on then another settings view of the desktop
        * was successfully opened
        */
    }
    else
    {
        /* WinOpenObject failed */
    }
}
```

```
}
```

WinOpenObject - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinOpenWindowDC

WinOpenWindowDC - Syntax

This function opens a device context for a window.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND     hwnd; /* Window handle. */
HDC      hdc;  /* Device-context handle. */

hdc = WinOpenWindowDC(hwnd);
```

WinOpenWindowDC Parameter - hwnd

hwnd ([HWND](#)) - input
Window handle.

WinOpenWindowDC Return Value - hdc

hdc ([HDC](#)) - returns
Device-context handle.

WinOpenWindowDC - Parameters

hwnd ([HWND](#)) - input
Window handle.

hdc ([HDC](#)) - returns
Device-context handle.

WinOpenWindowDC - Remarks

hdc is used to associate a presentation space with the window.

Note: The window device context is automatically closed when its associated window is destroyed. It must not be closed with the [DevCloseDC](#) call.

The visible region of the device context is updated automatically as windows are rearranged.

WinOpenWindowDC - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinOpenWindowDC - Related Functions

Related Functions

- [WinBeginPaint](#)
- [WinEnableWindowUpdate](#)
- [WinEndPaint](#)
- [WinExcludeUpdateRegion](#)
- [WinGetClipPS](#)
- [WinGetPS](#)
- [WinGetScreenPS](#)
- [WinInvalidateRect](#)
- [WinInvalidateRegion](#)
- [WinIsWindowShowing](#)
- [WinIsWindowVisible](#)
- [WinLockVisRegions](#)
- [WinOpenWindowDC](#)
- [WinQueryUpdateRect](#)
- [WinQueryUpdateRegion](#)
- [WinRealizePalette](#)
- [WinReleasePS](#)
- [WinShowWindow](#)
- [WinUpdateWindow](#)

- [WinValidateRect](#)
- [WinValidateRegion](#)

WinOpenWindowDC - Example Code

This example calls WinOpenWindowDC to open a device context for a window, the handle to which is then used to associate a presentation space with the window.

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#define INCL_GPICONTROL      /* Gpi Control Functions */
#include <os2.h>

HWND  hwnd;          /* window handle */
HPS    hps;          /* presentation-space handle */
SIZEL  pagesize={0L,0L}; /* Presentation page size */
HAB    hab;          /* Anchor-block handle */
HDC    hdc;          /* device-context handle */

case WM_CREATE:      /* Window just created */

    hdc = WinOpenWindowDC(hwnd); /* Open window device context */

    hps = GpiCreatePS(hab,          /* Create GPI PS and */
                      hdc,          /* associate with DC */
                      &pagesize,   /* default size */
                      PU_PELS |    /* pixel units */
                      GPIF_LONG |  /* 4-byte coordinates */
                      GPIA_ASSOC); /* associate with device */
```

WinOpenWindowDC - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinPeekMsg

WinPeekMsg - Syntax

This function inspects the thread's message queue and returns to the application with or without a message.


```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
PQMSG    pqmsg;        /* Message structure. */
HWND     hwndFilter;   /* Window filter. */
ULONG    ulFirst;      /* First message identity. */
ULONG    ulLast;       /* Last message identity. */
ULONG    flOptions;    /* Options. */
BOOL     rc;           /* Message-available indicator. */

rc = WinPeekMsg(hab, pqmsg, hwndFilter, ulFirst,
               ulLast, flOptions);
```

WinPeekMsg Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinPeekMsg Parameter - pqmsg

pqmsg (**PQMSG**) - output
Message structure.

Note: If the function returns FALSE, the state of the QMSG structure is undefined. Applications should always check the return code before examining this structure.

WinPeekMsg Parameter - hwndFilter

hwndFilter (**HWND**) - input
Window filter.

WinPeekMsg Parameter - ulFirst

ulFirst (**ULONG**) - input
First message identity.

WinPeekMsg Parameter - ulLast

ulLast (**ULONG**) - input
Last message identity.

WinPeekMsg Parameter - flOptions

flOptions (**ULONG**) - input
Options.

If neither of the following flags is specified, the message is not removed. If both of the following flags are specified, the message is removed:

PM_REMOVE	Remove message from queue
PM_NOREMOVE	Do not remove message from queue.

WinPeekMsg Return Value - rc

rc (**BOOL**) - returns
Message-available indicator.

TRUE	Message available
FALSE	No message available.

WinPeekMsg - Parameters

hab (**HAB**) - input
Anchor-block handle.

pqmsg (**PQMSG**) - output
Message structure.

Note: If the function returns FALSE, the state of the QMSG structure is undefined. Applications should always check the return code before examining this structure.

hwndFilter (**HWND**) - input
Window filter.

ulFirst (**ULONG**) - input
First message identity.

ulLast (**ULONG**) - input
Last message identity.

flOptions (**ULONG**) - input

Options.

If neither of the following flags is specified, the message is not removed. If both of the following flags are specified, the message is removed:

PM_REMOVE	Remove message from queue
PM_NOREMOVE	Do not remove message from queue.

rc ([BOOL](#)) - returns
Message-available indicator.

TRUE	Message available
FALSE	No message available.

WinPeekMsg - Remarks

This function is identical to the [WinGetMsg](#) function, except that it does not wait for the arrival of a message. The message can be left on the queue, by using */fOptions*.

For details of *hwndFilter*, *ulFirst*, and *ulLast*, see the [WinGetMsg](#) function.

The window handle within *pqmsg* is null if the message is posted to the queue with a *hwnd* that is null.

WinPeekMsg - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_INVALID_FLAG (0x1019)
An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

WinPeekMsg - Related Functions

Related Functions

- [WinBroadcastMsg](#)
- [WinCancelShutdown](#)
- [WinCreateMsgQueue](#)
- [WinDestroyMsgQueue](#)
- [WinDispatchMsg](#)
- [WinGetDlgMsg](#)
- [WinGetMsg](#)
- [WinInSendMessage](#)
- [WinPeekMsg](#)
- [WinPostMsg](#)
- [WinPostQueueMsg](#)
- [WinQueryMsgPos](#)
- [WinQueryMsgTime](#)
- [WinQueryQueueInfo](#)

- [WinQueryQueueStatus](#)
- [WinSendDlgItemMsg](#)
- [WinSendMsg](#)
- [WinSetClassMsgInterest](#)
- [WinSetMsgInterest](#)
- [WinSetMsgMode](#)
- [WinSetSynchroMode](#)
- [WinWaitMsg](#)

WinPeekMsg - Example Code

This example uses WinPeekMsg to count the total number of pending messages for the window corresponding to hwndFilter.

```
#define INCL_WINMESSAGEGR      /* Window Message Functions */
#define INCL_WINWINDOWMGR     /* Window Manager Functions */
#include <os2.h>

HAB      hab;           /* anchor-block handle */
QMSG     qmsg;          /* message */
HWND     hwndFilter;    /* message queue filter */
ULONG    flOptions;     /* peek options */
ULONG    ulMsgCount=0;  /* message count */

/* don't remove messages */
flOptions = PM_NOREMOVE;

/* count number of messages for filter window */
while (WinPeekMsg (hab, &qmsg, hwndFilter, 0, 0, flOptions))
    ulMsgCount++;
```

WinPeekMsg - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinPopupMenu

WinPopupMenu - Syntax

This function causes a pop-up menu to be presented.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndParent; /* Parent-window handle. */
HWND    hwndOwner;  /* Owner-window handle. */
HWND    hwndMenu;   /* Pop-up menu-window handle. */
LONG    x;          /* x-coordinate of the pop-up menu position. */
LONG    y;          /* y-coordinate of the pop-up menu position. */
LONG    idItem;     /* Item identity. */
ULONG   fs;         /* Options. */
BOOL    rc;         /* Pop-up menu invoked indicator. */

rc = WinPopupMenu(hwndParent, hwndOwner, hwndMenu,
    x, y, idItem, fs);
```

WinPopupMenu Parameter - hwndParent

hwndParent ([HWND](#)) - input
Parent-window handle.

WinPopupMenu Parameter - hwndOwner

hwndOwner ([HWND](#)) - input
Owner-window handle.

The owner window receives all the notification messages generated by the pop-up menu.

WinPopupMenu Parameter - hwndMenu

hwndMenu ([HWND](#)) - input
Pop-up menu-window handle.

The pop-up menu must have been created, by use of either the [WinCreateMenu](#) or [WinLoadMenu](#) functions.

WinPopupMenu Parameter - x

x ([LONG](#)) - input
x-coordinate of the pop-up menu position.

The value is in window coordinates relative to the origin of the parent window.

The x-coordinate of the origin of the pop-up menu can be affected, if either of the [PU_POSITIONONITEM](#) or [PU_HCONSTRAIN](#)

values of the *fs* parameter is also set.

WinPopupMenu Parameter - y

y (**LONG**) - input
y-coordinate of the pop-up menu position.

The value is in window coordinates relative to the origin of the parent window.

The y-coordinate of the origin of the pop-up menu can be affected, if either of the PU_POSITIONONITEM or PU_VCONSTRAIN values of the *fs* parameter is also set.

WinPopupMenu Parameter - idItem

idItem (**LONG**) - input
Item identity.

This is used if either the PU_POSITIONONITEM or PU_SELECTITEM of the *fs* parameter is also set. It must be greater or equal to 0 and less or equal to 0xFFFF.

WinPopupMenu Parameter - fs

fs (**ULONG**) - input
Options.

Position

Pop-up menu position.

PU_POSITIONONITEM

Position the pop-up menu so that the item identified by the *idItem* parameter of the top-level menu specified by the *hwndMenu* parameter lies directly under the *x* \ *y* coordinates.

The position of the pop-up menu can be affected, if either the PU_HCONSTRAIN or or PU_VCONSTRAIN values of the *fs* parameter is also set.

This value also causes the pop-up menu item identified by the *idItem* to be selected.

Restrain

Pop-up menu position constraints.

These options allow the application to ensure that the pop-up menu is visible on the desktop.

PU_HCONSTRAIN

Constrain the pop-up menu so that its width is wholly visible on the desktop.

If necessary the position of the pop-up menu will be adjusted so that its left edge is coincident with the left edge of the desktop or that its right edge is coincident with the right edge of the desktop.

PU_VCONSTRAIN

Constrain the pop-up menu so that its height is wholly visible on the desktop.

If necessary the position of the pop-up menu will be adjusted so that its top edge is coincident with the top edge of the desktop or that its bottom edge is coincident with the bottom edge of the desktop.

InitialState

Initial input state of the pop-up menu.

This allows the user interaction which caused the application to summon the pop-up menu to be carried through as the initial user interaction with the pop-up menu.

For example, this permits the application to support the user interface in which mouse button 1 can be depressed to cause the pop-up menu to be presented and held down while moving the mouse over the menu in order to select another menu item and then released to dismiss the menu.

Only one of the following values can be selected:

PU_MOUSEBUTTON1DOWN

The pop-up menu is initialized with mouse button 1 depressed.

PU_MOUSEBUTTON2DOWN

The pop-up menu is initialized with mouse button 2 depressed.

PU_MOUSEBUTTON3DOWN

The pop-up menu is initialized with mouse button 3 depressed.

PU_NONE

The pop-up menu is to be presented uninfluenced by the user interaction which caused it to be summoned.

This is the default value.

Select

Item selection.

PU_SELECTITEM

The item identified by *idItem* is to be selected. This is only valid if PU_NONE is set in the *InitialState* parameter.

If the identified item is in a submenu of the pop-up menu, then all the previous submenus in the menu hierarchy are presented with the correct path to the identified item.

Usage

Input device usage.

The window procedure controlling the pop-up menu must be informed of which input devices are available for interaction with the pop-up menu.

These options are independent to those of the *InitialState* parameter. Therefore, if an application indicates in the *InitialState* parameter that the pop-up menu is to be initialized with a particular user interaction, then the mechanism which permits that user interaction would usually be specified in this parameter. In this way the user's expectation, that once a device has been employed for the manipulation of the pop-up menu then that device can continue to be used for that purpose, is fulfilled.

It is valid to specify a user interaction as an initialization of the pop-up menu by an input mechanism which is not identified as available for interaction with the pop-up menu. This implies that the user cannot necessarily complete the interaction with the pop-up menu with that input mechanism.

For example, if a pop-up menu is initialized with a mouse button depressed but that mouse button is not identified as available for manipulating the pop-up menu, then that mouse button can manipulate the pop-up menu until it is released. Assuming that the pop-up menu is not dismissed when that mouse button is released, then the mouse button cannot be used for further interaction with the pop-up menu, since it is not identified as available for that use.

The following list shows the input device valid for interaction with the pop-up menu with each option:

PU_KEYBOARD

The keyboard.

PU_MOUSEBUTTON1

Mouse button 1.

PU_MOUSEBUTTON2

Mouse button 2.

PU_MOUSEBUTTON3

Mouse button 3.

WinPopupMenu Return Value - rc

rc (**BOOL**) - returns

Pop-up menu invoked indicator.

This function returns as soon as the pop-up menu has been invoked, which might be before the user has completed interacting with the pop-up menu.

TRUE

Pop-up menu successfully invoked

FALSE

Pop-up menu not successfully invoked.

WinPopupMenu - Parameters

hwndParent (**HWND**) - input

Parent-window handle.

hwndOwner (**HWND**) - input

Owner-window handle.

The owner window receives all the notification messages generated by the pop-up menu.

hwndMenu (**HWND**) - input

Pop-up menu-window handle.

The pop-up menu must have been created, by use of either the [WinCreateMenu](#) or [WinLoadMenu](#) functions.

x (**LONG**) - input

x-coordinate of the pop-up menu position.

The value is in window coordinates relative to the origin of the parent window.

The x-coordinate of the origin of the pop-up menu can be affected, if either of the `PU_POSITIONONITEM` or `PU_HCONSTRAIN` values of the `/s` parameter is also set.

y (**LONG**) - input

y-coordinate of the pop-up menu position.

The value is in window coordinates relative to the origin of the parent window.

The y-coordinate of the origin of the pop-up menu can be affected, if either of the `PU_POSITIONONITEM` or `PU_VCONSTRAIN` values of the `/s` parameter is also set.

idItem (**LONG**) - input

Item identity.

This is used if either the `PU_POSITIONONITEM` or `PU_SELECTITEM` of the `/s` parameter is also set. It must be greater or equal to 0 and less or equal to 0xFFFF.

fs (**ULONG**) - input

Options.

Position

Pop-up menu position.

PU_POSITIONONITEM

Position the pop-up menu so that the item identified by the *idItem* parameter of the top-level menu specified by the *hwndMenu* parameter lies directly under the $x \setminus y$ coordinates.

The position of the pop-up menu can be affected, if either the PU_HCONSTRAIN or or PU_VCONSTRAIN values of the *fs* parameter is also set.

This value also causes the pop-up menu item identified by the *idItem* to be selected.

Restrain

Pop-up menu position constraints.

These options allow the application to ensure that the pop-up menu is visible on the desktop.

PU_HCONSTRAIN

Constrain the pop-up menu so that its width is wholly visible on the desktop.

If necessary the position of the pop-up menu will be adjusted so that its left edge is coincident with the left edge of the desktop or that its right edge is coincident with the right edge of the desktop.

PU_VCONSTRAIN

Constrain the pop-up menu so that its height is wholly visible on the desktop.

If necessary the position of the pop-up menu will be adjusted so that its top edge is coincident with the top edge of the desktop or that its bottom edge is coincident with the bottom edge of the desktop.

InitialState

Initial input state of the pop-up menu.

This allows the user interaction which caused the application to summon the pop-up menu to be carried through as the initial user interaction with the pop-up menu.

For example, this permits the application to support the user interface in which mouse button 1 can be depressed to cause the pop-up menu to be presented and held down while moving the mouse over the menu in order to select another menu item and then released to dismiss the menu.

Only one of the following values can be selected:

PU_MOUSEBUTTON1DOWN

The pop-up menu is initialized with mouse button 1 depressed.

PU_MOUSEBUTTON2DOWN

The pop-up menu is initialized with mouse button 2 depressed.

PU_MOUSEBUTTON3DOWN

The pop-up menu is initialized with mouse button 3 depressed.

PU_NONE

The pop-up menu is to be presented uninfluenced by the user interaction which caused it to be summoned.

This is the default value.

Select

Item selection.

PU_SELECTITEM

The item identified by *idItem* is to be selected. This is only valid if PU_NONE is set in the *InitialState* parameter.

If the identified item is in a submenu of the pop-up menu, then all the previous submenus in the menu hierarchy are presented with the correct path to the identified item.

Usage

Input device usage.

The window procedure controlling the pop-up menu must be informed of which input devices are available for interaction with the pop-up menu.

These options are independent to those of the *InitialState* parameter. Therefore, if an application indicates in the *InitialState* parameter that the pop-up menu is to be initialized with a particular user interaction, then the

mechanism which permits that user interaction would usually be specified in this parameter. In this way the user's expectation, that once a device has been employed for the manipulation of the pop-up menu then that device can continue to be used for that purpose, is fulfilled.

It is valid to specify a user interaction as an initialization of the pop-up menu by an input mechanism which is not identified as available for interaction with the pop-up menu. This implies that the user cannot necessarily complete the interaction with the pop-up menu with that input mechanism.

For example, if a pop-up menu is initialized with a mouse button depressed but that mouse button is not identified as available for manipulating the pop-up menu, then that mouse button can manipulate the pop-up menu until it is released. Assuming that the pop-up menu is not dismissed when that mouse button is released, then the mouse button cannot be used for further interaction with the pop-up menu, since it is not identified as available for that use.

The following list shows the input device valid for interaction with the pop-up menu with each option:

PU_KEYBOARD	The keyboard.
PU_MOUSEBUTTON1	Mouse button 1.
PU_MOUSEBUTTON2	Mouse button 2.
PU_MOUSEBUTTON3	Mouse button 3.

rc ([BOOL](#)) - returns

Pop-up menu invoked indicator.

This function returns as soon as the pop-up menu has been invoked, which might be before the user has completed interacting with the pop-up menu.

TRUE

Pop-up menu successfully invoked

FALSE

Pop-up menu not successfully invoked.

WinPopupMenu - Remarks

A pop-up menu is the unanchored equivalent of a pull-down menu, that is it can be positioned anywhere rather than being associated with an action bar. Typically, pop-up menus are related to specific objects, such as an icon, or with a particular area of the application's presentation space.

Once invoked, a pop-up menu behaves in exactly the same way as a pull-down menu.

WinPopupMenu - Related Functions

Related Functions

- [WinCreateMenu](#)
- [WinLoadMenu](#)
- [WinPopupMenu](#)

WinPopupMenu - Example Code

This example presents a pop-up menu (loaded from RES.DLL by WinLoadMenu) with the following characteristics: located at (0,50);

initialized with mouse button 1 depressed; allowing keyboard and mouse button 1 interaction.

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#define INCL_WINMENUS         /* Window Menu Functions */
#include <os2.h>

HWND      hwndMenu;           /* menu window */
HWND      hwndOwner;          /* owner window */
HMODULE    hmodDLL;           /* resource handle */
ULONG      idMenuid;          /* resource menu id */
BOOL       fSuccess;          /* success indicator */
HWND      hwndParent;         /* parent window */
ULONG      flOptions;         /* pop-up menu options */

if (DosQueryModuleHandle("RES.DLL",&hmodDLL))
    hwndMenu = WinLoadMenu(hwndOwner, hmodDLL, idMenuid);

flOptions = PU_MOUSEBUTTON1DOWN | PU_KEYBOARD | PU_MOUSEBUTTON1;
fSuccess = WinPopupMenu(hwndParent, hwndOwner, hwndMenu, 0, 50,
                        0, flOptions);
```

WinPopupMenu - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinPostMsg

WinPostMsg - Syntax

This function posts a message to the message queue associated with the window defined by *hwnd*.

```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND      hwnd;           /* Window handle. */
ULONG      ulMsgid;       /* Message identity. */
MPARAM      mpParam1;     /* Parameter 1. */
MPARAM      mpParam2;     /* Parameter 2. */
BOOL       rc;            /* Message-posted indicator. */

rc = WinPostMsg(hwnd, ulMsgid, mpParam1, mpParam2);
```

WinPostMsg Parameter - hwnd

hwnd (HWND) - input	Window handle.
NULL	The message is posted into the queue associated with the current thread. When the message is received by using the WinGetMsg or WinPeekMsg functions, the parameter of the QMSG structure is NULL.
Other	Window handle.

WinPostMsg Parameter - ulMsgid

ulMsgid (ULONG) - input	Message identity.
--	-------------------

WinPostMsg Parameter - mpParam1

mpParam1 (MPARAM) - input	Parameter 1.
--	--------------

WinPostMsg Parameter - mpParam2

mpParam2 (MPARAM) - input	Parameter 2.
--	--------------

WinPostMsg Return Value - rc

rc (BOOL) - returns	Message-posted indicator.
TRUE	Message successfully posted
FALSE	Message could not be posted; for example, because the message queue was full.

WinPostMsg - Parameters

hwnd ([HWND](#)) - input
Window handle.

NULL

The message is posted into the queue associated with the current thread. When the message is received by using the [WinGetMsg](#) or [WinPeekMsg](#) functions, the parameter of the [QMSG](#) structure is NULL.

Other

Window handle.

ulMsgid ([ULONG](#)) - input
Message identity.

mpParam1 ([MPARAM](#)) - input
Parameter 1.

mpParam2 ([MPARAM](#)) - input
Parameter 2.

rc ([BOOL](#)) - returns
Message-posted indicator.

TRUE

Message successfully posted

FALSE

Message could not be posted; for example, because the message queue was full.

WinPostMsg - Remarks

The message contains *hwnd*, *ulMsgid*, *mpParam1*, *mpParam2*, and the time and pointer position when this function is called.

WinPostMsg returns immediately, while [WinSendMsg](#) waits for the receiver to return.

A thread which does not have a message queue can still call WinPostMsg but cannot call [WinSendMsg](#).

WinPostMsg - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinPostMsg - Related Functions

Related Functions

- [WinBroadcastMsg](#)
- [WinCreateMsgQueue](#)

- [WinDestroyMsgQueue](#)
- [WinDispatchMsg](#)
- [WinGetDlgMsg](#)
- [WinGetMsg](#)
- [WinInSendMessage](#)
- [WinPeekMsg](#)
- [WinPostMsg](#)
- [WinPostQueueMsg](#)
- [WinQueryMsgPos](#)
- [WinQueryMsgTime](#)
- [WinQueryQueueInfo](#)
- [WinQueryQueueStatus](#)
- [WinSendDlgItemMsg](#)
- [WinSendMessage](#)
- [WinSetClassMsgInterest](#)
- [WinSetMsgInterest](#)
- [WinSetMsgMode](#)
- [WinSetSynchroMode](#)
- [WinWaitMsg](#)

WinPostMsg - Example Code

This example posts a Set menu item checked attribute message (MM_SETITEMATTR) to the message queue associated with the window handle in response to a menu select message (WM_MENUSELECT).

```
#define INCL_WINMESSAGEGR      /* Window Message Functions */
#define INCL_WINMENUS         /* Window Menu Functions */
#include <os2.h>

BOOL      fResult;           /* message-posted indicator */
ULONG     ulMsgid;           /* message id */
MPARAM    mp1;              /* Parameter 1 (rectl structure) */
MPARAM    mp2;              /* Parameter 2 (frame boolean) */
USHORT     usItemId;         /* menu item id */
HWND      hwndMenu;         /* menu handle */

case WM_MENUSELECT:
    usItemId = SHORT1FROMMP(mp1);
    hwndMenu = HWNDFROMMP(mp2);

    /* initialize message id, parameters */
    ulMsgid = MM_SETITEMATTR;
    mp1 = MPFROM2SHORT(usItemId, TRUE);
    mp2 = MPFROM2SHORT(MIA_CHECKED, TRUE);

    fResult = WinPostMsg(hwndMenu, ulMsgid, mp1, mp2);
```

WinPostMsg - Topics

Select an item:

- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Errors](#)
 - [Remarks](#)
 - [Example Code](#)
 - [Related Functions](#)
 - [Glossary](#)
-

WinPostQueueMsg

WinPostQueueMsg - Syntax

This function posts a message to a message queue.

```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HMQ      hmq; /* Message-queue handle. */
ULONG    msg; /* Message identifier. */
MPARAM    mp1; /* Parameter 1. */
MPARAM    mp2; /* Parameter 2. */
BOOL      rc; /* Success indicator. */

rc = WinPostQueueMsg(hmq, msg, mp1, mp2);
```

WinPostQueueMsg Parameter - hmq

hmq ([HMQ](#)) - input
Message-queue handle.

WinPostQueueMsg Parameter - msg

msg ([ULONG](#)) - input
Message identifier.

WinPostQueueMsg Parameter - mp1

mp1 ([MPARAM](#)) - input
Parameter 1.

WinPostQueueMsg Parameter - mp2

mp2 ([MPARAM](#)) - input
Parameter 2.

WinPostQueueMsg Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred, or the queue was full.

WinPostQueueMsg - Parameters

hmq ([HMQ](#)) - input
Message-queue handle.

msg ([ULONG](#)) - input
Message identifier.

mp1 ([MPARAM](#)) - input
Parameter 1.

mp2 ([MPARAM](#)) - input
Parameter 2.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred, or the queue was full.

WinPostQueueMsg - Remarks

This function can be used to post messages to any queue in the system.

It constructs a [QMSG](#) structure by setting its parameters from the corresponding parameters of this function, and by deriving its parameters from the system time and pointer position when this function was called. The [QMSG](#) structure is then placed on the specified queue.

WinPostQueueMsg - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HMQ (0x1002)
An invalid message-queue handle was specified.

WinPostQueueMsg - Related Functions

Related Functions

- [WinBroadcastMsg](#)
 - [WinCreateMsgQueue](#)
 - [WinDestroyMsgQueue](#)
 - [WinDispatchMsg](#)
 - [WinGetDlgMsg](#)
 - [WinGetMsg](#)
 - [WinInSendMessage](#)
 - [WinPeekMsg](#)
 - [WinPostMsg](#)
 - [WinPostQueueMsg](#)
 - [WinQueryMsgPos](#)
 - [WinQueryMsgTime](#)
 - [WinQueryQueueInfo](#)
 - [WinQueryQueueStatus](#)
 - [WinSendDlgItemMsg](#)
 - [WinSendMessage](#)
 - [WinSetClassMsgInterest](#)
 - [WinSetMsgInterest](#)
 - [WinSetMsgMode](#)
 - [WinSetSynchroMode](#)
 - [WinWaitMsg](#)
-

WinPostQueueMsg - Example Code

This example posts a Set menu item checked attribute message (MM_SETITEMATTR) to the specified message queue in response to a menu select message (WM_MENUSELECT).

```
#define INCL_WINMESSAGEMGR      /* Window Message Functions */
#define INCL_WINMENUMS        /* Window Menu Functions */
#include <os2.h>

BOOL      fResult;              /* message-posted indicator */
ULONG     ulMsgid;              /* message id */
HMQ       hmq;                 /* message queue handle */
MPARAM    mp1;                 /* Parameter 1 (rectl structure) */
MPARAM    mp2;                 /* Parameter 2 (frame boolean) */
USHORT    usItemId;            /* menu item id */

case WM_MENUSELECT:
    usItemId = SHORT1FROMMP(mp1);

    /* initialize message id, parameters */
    ulMsgid = MM_SETITEMATTR;
    mp1 = MPFROM2SHORT(usItemId, TRUE);
    mp2 = MPFROM2SHORT(MIA_CHECKED, TRUE);

    fResult = WinPostQueueMsg(hmq, ulMsgid, mp1, mp2);
```

WinPostQueueMsg - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinPrevChar

WinPrevChar - Syntax

This function moves to the previous character in a string.

```
#define INCL_WINCOUNTRY /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;           /* Anchor-block handle. */
ULONG    ulCodepage;    /* Code page. */
ULONG    ulCountry;     /* Reserved value, must be 0. */
PSZ      pszStart;      /* Character string that contains pszCurrentChar. */
PSZ      pszCurrentChar; /* Current character. */
PSZ      pszPrevChar;   /* Previous character. */

pszPrevChar = WinPrevChar(hab, ulCodepage,
                          ulCountry, pszStart, pszCurrentChar);
```

WinPrevChar Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinPrevChar Parameter - ulCodepage

ulCodepage ([ULONG](#)) - input
Code page.

If a non-existent code page is specified for *ulCodepage*, the code page specified by the queue associated with the calling thread is used.

WinPrevChar Parameter - ulCountry

ulCountry ([ULONG](#)) - input
Reserved value, must be 0.

WinPrevChar Parameter - pszStart

pszStart ([PSZ](#)) - input
Character string that contains *pszCurrentChar*.

WinPrevChar Parameter - pszCurrentChar

pszCurrentChar ([PSZ](#)) - input
Current character.

WinPrevChar Return Value - pszPrevChar

pszPrevChar ([PSZ](#)) - returns
Previous character.

The previous character, or the first character if *pszCurrentChar* is the first character of *pszStart*.

WinPrevChar - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

ulCodepage ([ULONG](#)) - input
Code page.

If a non-existent code page is specified for *ulCodepage*, the code page specified by the queue associated with the calling thread is used.

ulCountry ([ULONG](#)) - input
Reserved value, must be 0.

pszStart ([PSZ](#)) - input

Character string that contains *pszCurrentChar*.

pszCurrentChar (PSZ) - input
Current character.

pszPrevChar (PSZ) - returns
Previous character.

The previous character, or the first character if *pszCurrentChar* is the first character of *pszStart*.

WinPrevChar - Remarks

This function handles DBCS strings.

The following is the list of valid code pages:

Country	Code Page
Canadian-French	863
Desktop Publishing	1004
Iceland	861
Latin 1 Multilingual	850
Latin 2 Multilingual	852
Nordic	865
Portuguese	860
Turkey	857
U.S. (IBM PC)	437

Code page 1004 is compatible with Microsoft** Windows**.

The following EBCDIC code pages, based on character set 697, are also available for output:

Country	Code Page
Austrian/German	273
Belgian	500
Brazil	037
Czechoslovakia	870
Danish/Norwegian	277
Finnish/Swedish	278
French	297
Hungary	870
Iceland	871
International	500
Italian	280
Poland	870
Portuguese	037
Spanish	284
Turkey	1026
U.K.-English	285
U.S.-English	037
Yugoslavia	870

Code pages 274 (Belgian) and 282 (Portuguese) can be used to provide access to old data. The following is the list of valid country codes:

Country	Code
Arabic	785
Australian	61
Belgian	32
Canadian-French	2
Danish	45
Finnish	358
French	33
German	49
Hebrew	972
Italian	39
Japanese	81
Korean	82


```

COUNTRYCODE Country;
COUNTRYINFO CtryBuffer;

Country.country = CURRENT_COUNTRY;

DosQueryCp((ULONG)2,
            &CodePage,           /* Get code page identifier */
            &DataLength);

/* First WORD contains the codepage */
Country.codepage = (ULONG)HIUSHORT(CodePage);

/* Get the corresponding country code */
DosQueryCtryInfo(sizeof(CtryBuffer), /* Length of data area */
                  &Country,          /* Input data structure */
                  &CtryBuffer,       /* Data area to be filled */
                  &DataLength);      /* Length of data */

/* A pointer should be returned to character "D" in the string */
ptoD = WinPrevChar(hab,
                  (ULONG)CodePage,
                  (ULONG)CtryBuffer.country,
                  (PSZ)string,
                  ptoE); /* Pointer to character "E" */

printf(ptoE);
}

```

WinPrevChar - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinProcessDlg

WinProcessDlg - Syntax

This function dispatches messages while a modal dialog window is displayed.

```

#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndDlg; /* Dialog-window handle. */
ULONG    ulReply; /* Reply value. */

ulReply = WinProcessDlg(hwndDlg);

```

WinProcessDlg Parameter - hwndDlg

hwndDlg ([HWND](#)) - input
Dialog-window handle.

WinProcessDlg Return Value - ulReply

ulReply ([ULONG](#)) - returns
Reply value.

Value established by the [WinDismissDlg](#) function.

WinProcessDlg - Parameters

hwndDlg ([HWND](#)) - input
Dialog-window handle.

ulReply ([ULONG](#)) - returns
Reply value.

Value established by the [WinDismissDlg](#) function.

WinProcessDlg - Remarks

If the dialog has an owner window, that window is disabled. This means that all user input to the owner, and its descendants, is prevented.

This function then dispatches messages from the queue to the appropriate window or dialog procedure until the dialog is dismissed by [WinDismissDlg](#). This is usually done by the dialog procedure on receipt of an appropriate message, but also occurs if the dialog procedure passes a [WM_COMMAND](#) message to [WinDefDlgProc](#) or if the [WM_QUIT](#) message is encountered before the dialog window is dismissed. In this latter case, [WinProcessDlg](#) itself issues [WinDismissDlg](#), and posts the [WM_QUIT](#) message back to the queue so that the application main loop terminates in the normal way.

This function shows the window, if it is hidden, when the queue is empty. It is therefore possible for the experienced user to type ahead and cause the dialog to be dismissed before it becomes visible.

If the dialog window needs to be processed through [WinProcessDlg](#), after being dismissed, the [FF_DLGDISMISSED](#) flag must be reset.

[WinDismissDlg](#) hides the dialog window without destroying it, and also re-enables any window that was disabled by this function.

This function does not return until a [WinDismissDlg](#) call is issued in one of the ways listed above. This is true even if the application main window has not been disabled, for example because the dialog window has no owner. In this case, the dialog will appear to the user to be modeless; the user will continue to be able to interact with the application, and possibly create multiple instances of the dialog. In such circumstances the operating system calls the application main window procedure recursively before [WinProcessDlg](#) returns.

It is not possible to temporarily disable more than one window using this function; a dialog window can have at most one owner. If an application has more than one main window which should be disabled while the modal dialog is displayed, it can be done by setting appropriate hooks using [WinSetHook](#).

If the dialog window is a descendant of its owner, this function disables input to the dialog itself. However, this situation can only occur by explicitly changing the window hierarchy. Dialog windows are created using [WinLoadDlg](#) or [WinCreateDlg](#), which modify the owner window specified on their parameter lists.

WinProcessDlg - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinProcessDlg - Related Functions

Related Functions

- [WinCreateDlg](#)
- [WinDefDlgProc](#)
- [WinDismissDlg](#)
- [WinDlgBox](#)
- [WinGetDlgMsg](#)
- [WinLoadDlg](#)
- [WinProcessDlg](#)

WinProcessDlg - Related Messages

Related Messages

- [WM_COMMAND](#)
- [WM_QUIT](#)

WinProcessDlg - Example Code

This function is used to process messages while a dialog is active.

```
#include <os2.h>

#define INCL_WIN
#define INCL_WINDIALOGS
#define IDD_OPEN WM_USER+200
#define IDM_OPEN WM_USER+201

HWND hwndDlg;
HWND hwndFrame;
PFNWP OpenDlg;
```



```

/* Inside client procedure */
switch(msg)
{
    case WM_COMMAND:
        /* The user has chosen a menu item. */
        /* Process the selection accordingly. */
        switch (SHORT1FROMMP(mpl))
        {
            case IDM_OPEN:
                if (WinDlgBox(HWND_DESKTOP,
                             hwndFrame, /* Handle of the owner */
                             OpenDlg, /* Dialog procedure address */
                             (ULONG)0, /* Location of dialog resource */
                             IDD_OPEN, /* Resource identifier */
                             NULL)) /* Application-specific data */
                {
                    WinProcessDlg(hwndDlg);
                }
                break;
        }
        break;
}

```

WinProcessDlg - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinPtInRect

WinPtInRect - Syntax

This function queries whether a point lies within a rectangle.

```

#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab; /* Anchor-block handle. */
PRECTL   prcl; /* Rectangle to be queried. */
PPOINTL  pptl; /* Point to be queried. */
BOOL     rc; /* Success indicator. */

rc = WinPtInRect(hab, prcl, pptl);

```

WinPtInRect Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinPtInRect Parameter - prcl

prcl ([PRECTL](#)) - input
Rectangle to be queried.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) may also be used, if supported by the language.

WinPtInRect Parameter - pptl

pptl ([PPOINTL](#)) - input
Point to be queried.

WinPtInRect Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

- | | |
|-------|---|
| TRUE | <i>pptl</i> lies within <i>prcl</i> . |
| FALSE | <i>pptl</i> does not lie within <i>prcl</i> , or an error occurred. |
-

WinPtInRect - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

prcl ([PRECTL](#)) - input
Rectangle to be queried.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) may also be used, if supported by the language.

pptl ([PPOINTL](#)) - input
Point to be queried.

rc ([BOOL](#)) - returns
Success indicator.

TRUE
FALSE

pptl lies within *prcl*.

pptl does not lie within *prcl*, or an error occurred.

WinPtInRect - Related Functions

Related Functions

- [WinCopyRect](#)
- [WinEqualRect](#)
- [WinFillRect](#)
- [WinInflateRect](#)
- [WinIntersectRect](#)
- [WinIsRectEmpty](#)
- [WinOffsetRect](#)
- [WinPtInRect](#)
- [WinSetRect](#)
- [WinSetRectEmpty](#)
- [WinSubtractRect](#)
- [WinUnionRect](#)

WinPtInRect - Example Code

This example processes a WM_BUTTON1UP message, converts the mouse pointer coordinates into a POINTL structure, and calls WinPtInRect to determine if the mouse was clicked in the predefined global rectangle.

```
#define INCL_WIN
#define INCL_WINRECTANGLES
#include <OS2.H>

HAB hab;          /* anchor-block handle */
/* . */
/* . */
RECTL rclGlobal;
POINTL ptl;
HPS hps;

USHORT msg;
MPARAM mpl;

/* inside client window function. */

switch(msg)
{
case WM_COMMAND:
/* The user has chosen a menu item. Process the selection */
/* accordingly. */
```

```

switch ( SHORT1FROMMP( mp1 ) )
{

case WM_BUTTON1UP:
    ptl.x = (LONG) SHORT1FROMMP(mp1);
    ptl.y = (LONG) SHORT2FROMMP(mp1);
    WinPtInRect(hab,      /* anchor-block handle      */
        &rclGlobal,      /* address of the rectangle */
        &ptl);          /* address of the point     */
    break;
}
break;
}

```

WinPtInRect - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinQueryAccelTable

WinQueryAccelTable - Syntax

This function queries the window or queue accelerator table.

```

#define INCL_WINACCELERATORS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
HWND     hwndFrame;     /* Frame-window handle. */
HACCEL    haccelAccel;  /* Accelerator-table handle. */

haccelAccel = WinQueryAccelTable(hab, hwndFrame);

```

WinQueryAccelTable Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinQueryAccelTable Parameter - hwndFrame

hwndFrame ([HWND](#)) - input
Frame-window handle.

NULLHANDLE

Return queue accelerator.

Other

Return the window accelerator table, by sending the [WM_QUERYACCELTABLE](#) message to *hwndFrame*.

WinQueryAccelTable Return Value - haccelAccel

haccelAccel ([HACCEL](#)) - returns
Accelerator-table handle.

NULLHANDLE

Error occurred

Other

Accelerator-table handle.

WinQueryAccelTable - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

hwndFrame ([HWND](#)) - input
Frame-window handle.

NULLHANDLE

Return queue accelerator.

Other

Return the window accelerator table, by sending the [WM_QUERYACCELTABLE](#) message to *hwndFrame*.

haccelAccel ([HACCEL](#)) - returns
Accelerator-table handle.

NULLHANDLE

Error occurred

Other

Accelerator-table handle.

WinQueryAccelTable - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)

An invalid window handle was specified.

WinQueryAccelTable - Related Functions

Related Functions

- [WinCopyAccelTable](#)
 - [WinCreateAccelTable](#)
 - [WinDestroyAccelTable](#)
 - [WinLoadAccelTable](#)
 - [WinQueryAccelTable](#)
 - [WinSetAccelTable](#)
 - [WinTranslateAccel](#)
-

WinQueryAccelTable - Related Messages

Related Messages

- [WM_QUERYACCELTABLE](#)
-

WinQueryAccelTable - Example Code

This example shows how to get the accelerator table for the frame window.

```
#define INCL_WINWINDOWMGR
#define INCL_WINACCELERATORS
#include <OS2.H>

HACCEL haccel;
HWND hwndFrame, hwndClient; /* window handles. */
HAB hab; /* anchor block. */

hwndFrame = WinQueryWindow(hwndClient,
                           QW_PARENT); /* get handle of parent, */
                                         /* which is frame window. */

/* Now get the accel table for the frame window */
haccel = WinQueryAccelTable(hab,
                           hwndFrame);
```

WinQueryAccelTable - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)

[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinQueryActiveDesktopPathname

WinQueryActiveDesktopPathname - Syntax

This function is specific to OS/2 Version 3.0 or higher.

This function returns the directory specification of the active desktop.

```
#define INCL_WINWORKPLACE
#include <os2.h>

PSZ      pszPathName; /* Memory allocated by caller in which directory specification is written. */
ULONG    ulSize;      /* Number of bytes pointed to by pszPathName. */
BOOL     rc;          /* Success indicator. */

rc = WinQueryActiveDesktopPathname(pszPathName,
                                   ulSize);
```

WinQueryActiveDesktopPathname Parameter - pszPathName

pszPathName ([PSZ](#)) - output
Memory allocated by caller in which directory specification is written.

WinQueryActiveDesktopPathname Parameter - ulSize

ulSize ([ULONG](#)) - input
Number of bytes pointed to by *pszPathName*.

WinQueryActiveDesktopPathname Return Value - rc

rc ([BOOL](#)) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinQueryActiveDesktopPathname - Parameters

pszPathName ([PSZ](#)) - output

Memory allocated by caller in which directory specification is written.

ulSize ([ULONG](#)) - input

Number of bytes pointed to by *pszPathName*.

rc ([BOOL](#)) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinQueryActiveDesktopPathname - Remarks

This function is used to find the directory specification of the current desktop. The current desktop is not always \DESKTOP of the boot drive.

WinQueryActiveDesktopPathname - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_PARAMETER (0x1645)

An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32,768 to +32,767 cannot be converted to a SHORT, and a negative number cannot be converted to a ULONG or USHORT.

WinQueryActiveDesktopPathname - Example Code

This example finds the directory specification of the current desktop.

```
#define INCL_WINWORKPLACE
#include <os2.h>

CHAR    szPath[CCHMAXPATH + 1];
BOOL    fSuccess;

fSuccess = WinQueryActiveDesktopPathname(szPath, sizeof(szPath));
if (fSuccess)
```



```

{
    /* WinQueryActiveDesktopPathname was successful */
}
else
{
    /* WinQueryActiveDesktopPathname failed */
}

```

WinQueryActiveDesktopPathname - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

WinQueryActiveWindow

WinQueryActiveWindow - Syntax

This function returns the active window for HWND_DESKTOP, or other parent window.

```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndParent; /* Parent-window handle for which the active window is required. */
HWND    hwndActive; /* Active-window handle. */

hwndActive = WinQueryActiveWindow(hwndParent);

```

WinQueryActiveWindow Parameter - hwndParent

hwndParent (**HWND**) - input

Parent-window handle for which the active window is required.

HWND_DESKTOP

The desktop-window handle that causes this function to return the top-level frame window.

Other

Specified parent-window handle.

WinQueryActiveWindow Return Value - hwndActive

hwndActive ([HWND](#)) - returns
Active-window handle.

NULLHANDLE
No window is active
Other
Active-window handle.

WinQueryActiveWindow - Parameters

hwndParent ([HWND](#)) - input
Parent-window handle for which the active window is required.

HWND_DESKTOP
The desktop-window handle that causes this function to return the top-level frame window.
Other
Specified parent-window handle.

hwndActive ([HWND](#)) - returns
Active-window handle.

NULLHANDLE
No window is active
Other
Active-window handle.

WinQueryActiveWindow - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinQueryActiveWindow - Related Functions

Related Functions

- [WinGetMinPosition](#)
- [WinQueryActiveWindow](#)
- [WinQueryWindowPos](#)
- [WinSaveWindowPos](#)
- [WinSetActiveWindow](#)
- [WinSetMultWindowPos](#)
- [WinSetWindowPos](#)

WinQueryActiveWindow - Example Code

This example shows how the WinQueryActiveWindow can be used to find the active window.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HWND hwndActive;

hwndActive = WinQueryActiveWindow(HWND_DESKTOP)
```

WinQueryActiveWindow - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Errors](#)
- [Example Code](#)
- [Related Functions](#)
- [Glossary](#)

WinQueryAnchorBlock

WinQueryAnchorBlock - Syntax

This function returns the anchor block handle of the caller.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND    hwnd; /* Window handle. */
HAB     hab;  /* Anchor block handle. */

hab = WinQueryAnchorBlock(hwnd);
```

WinQueryAnchorBlock Parameter - hwnd

hwnd ([HWND](#)) - input
Window handle.

WinQueryAnchorBlock Return Value - hab

hab ([HAB](#)) - returns
Anchor block handle.

NULLHANDLE

Invalid *hwnd* parameter

Other

Anchor block handle.

WinQueryAnchorBlock - Parameters

hwnd ([HWND](#)) - input
Window handle.

hab ([HAB](#)) - returns
Anchor block handle.

NULLHANDLE

Invalid *hwnd* parameter

Other

Anchor block handle.

WinQueryAnchorBlock - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinQueryAnchorBlock - Example Code

This function obtains the anchor block handle of the caller.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HAB hab;
HWND hwnd;

hab = WinQueryAnchorBlock(hwnd);
```

WinQueryAnchorBlock - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Glossary](#)

WinQueryAtomLength

WinQueryAtomLength - Syntax

This function queries the length of an atom represented by the specified atom.

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HATOMTBL    hatomtblAtomTbl; /* Atom-table handle. */
ATOM        atom;           /* Atom whose associated character-string length is to be returned. */
ULONG       ulretlen;        /* String length. */

ulretlen = WinQueryAtomLength(hatomtblAtomTbl,
                              atom);
```

WinQueryAtomLength Parameter - hatomtblAtomTbl

hatomtblAtomTbl ([HATOMTBL](#)) - input
Atom-table handle.

The handle returned from a previous [WinCreateAtomTable](#) or [WinQuerySystemAtomTable](#) function.

WinQueryAtomLength Parameter - atom

atom ([ATOM](#)) - input
Atom whose associated character-string length is to be returned.

WinQueryAtomLength Return Value - ulretlen

ulretlen ([ULONG](#)) - returns
String length.

0

The specified atom or the atom table is invalid.

Other

The length of the character string associated with the atom **excluding** the null terminating byte. Integer atoms always return a length of six.

WinQueryAtomLength - Parameters

hatomtblAtomTbl ([HATOMTBL](#)) - input
Atom-table handle.

The handle returned from a previous [WinCreateAtomTable](#) or [WinQuerySystemAtomTable](#) function.

atom ([ATOM](#)) - input
Atom whose associated character-string length is to be returned.

ulretlen ([ULONG](#)) - returns
String length.

0

The specified atom or the atom table is invalid.

Other

The length of the character string associated with the atom **excluding** the null terminating byte. Integer atoms always return a length of six.

WinQueryAtomLength - Remarks

The purpose of this function is to allow an application to determine the size of buffer to use in the [WinQueryAtomName](#) call.

WinQueryAtomLength - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HATOMTBL (0x1013)
An invalid atom-table handle was specified.

PMERR_INVALID_ATOM (0x1014)
The specified atom does not exist in the atom table.

WinQueryAtomLength - Related Functions

Related Functions

- [WinAddAtom](#)
- [WinCreateAtomTable](#)
- [WinDeleteAtom](#)
- [WinDestroyAtomTable](#)
- [WinFindAtom](#)
- [WinQueryAtomLength](#)
- [WinQueryAtomName](#)
- [WinQueryAtomUsage](#)
- [WinQuerySystemAtomTable](#)

WinQueryAtomLength - Example Code

This function queries the length of an atom.

```
#define INCL_WINATOM
#include <OS2.H>

HATOMTBL atomtbl;
ATOM atom = 25;

WinQueryAtomlength(atomtbl, /* atom handle. */
                    atom);
```

WinQueryAtomLength - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryAtomName

WinQueryAtomName - Syntax

This function returns an atom name associated with an atom.

```

#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HATOMTBL    hatomtblAtomTbl; /* Atom-table handle. */
ATOM        atom;           /* Identifies the character string to be retrieved. */
PSZ         pszBuffer;      /* Buffer to receive the character string. */
ULONG       ulBufferMax;     /* Buffer size in bytes. */
ULONG       ulretlen;        /* Length of retrieved character string. */

ulretlen = WinQueryAtomName(hatomtblAtomTbl,
                           atom, pszBuffer, ulBufferMax);

```

WinQueryAtomName Parameter - hatomtblAtomTbl

hatomtblAtomTbl ([HATOMTBL](#)) - input
Atom-table handle.

The handle returned from a previous [WinCreateAtomTable](#) or [WinQuerySystemAtomTable](#) function.

WinQueryAtomName Parameter - atom

atom ([ATOM](#)) - input
Identifies the character string to be retrieved.

WinQueryAtomName Parameter - pszBuffer

pszBuffer ([PSZ](#)) - output
Buffer to receive the character string.

WinQueryAtomName Parameter - ulBufferMax

ulBufferMax ([ULONG](#)) - input
Buffer size in bytes.

WinQueryAtomName Return Value - ulretlen

ulretlen ([ULONG](#)) - returns
Length of retrieved character string.

0
The specified atom or the atom table is invalid.

Other
The number of bytes copied to the buffer **excluding** the terminating zero.

WinQueryAtomName - Parameters

hatomtblAtomTbl ([HATOMTBL](#)) - input
Atom-table handle.

The handle returned from a previous [WinCreateAtomTable](#) or [WinQuerySystemAtomTable](#) function.

atom ([ATOM](#)) - input
Identifies the character string to be retrieved.

pszBuffer ([PSZ](#)) - output
Buffer to receive the character string.

ulBufferMax ([ULONG](#)) - input
Buffer size in bytes.

ulretlen ([ULONG](#)) - returns
Length of retrieved character string.

0
The specified atom or the atom table is invalid.

Other
The number of bytes copied to the buffer **excluding** the terminating zero.

WinQueryAtomName - Remarks

For integer atoms, the format of the string is "#dddd" where "dddd" are decimal digits in the system code page (an ASCII code page). No leading zeros are generated, and the length can be from 3 through 7 characters.

WinQueryAtomName - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HATOMTBL (0x1013)
An invalid atom-table handle was specified.

PMERR_INVALID_ATOM (0x1014)
The specified atom does not exist in the atom table.

PMERR_INVALID_STRING_PARM (0x100B)
The specified string parameter is invalid.

WinQueryAtomName - Related Functions

Related Functions

- [WinAddAtom](#)
- [WinCreateAtomTable](#)
- [WinDeleteAtom](#)
- [WinDestroyAtomTable](#)
- [WinFindAtom](#)
- [WinQueryAtomLength](#)
- [WinQueryAtomName](#)
- [WinQueryAtomUsage](#)
- [WinQuerySystemAtomTable](#)

WinQueryAtomName - Example Code

This function obtains the name of an atom given the atom id.

```
#define INCL_WINATOM
#include <OS2.H>
HATOMTBL atomtbl;
char atomname[256];
ATOM atom = 25;

WinQueryAtomName(atomtbl,
                  atom,
                  atomname,
                  sizeof(atomname));
```

WinQueryAtomName - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryAtomUsage

WinQueryAtomUsage - Syntax

This function returns the number of times an atom has been used.

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HATOMTBL    hatomtblAtomTbl; /* Atom-table handle. */
ATOM        atom;           /* Atom whose use count is to be returned. */
ULONG       ulcount;        /* Use count of the atom. */

ulcount = WinQueryAtomUsage(hatomtblAtomTbl,
                           atom);
```

WinQueryAtomUsage Parameter - hatomtblAtomTbl

hatomtblAtomTbl ([HATOMTBL](#)) - input
Atom-table handle.

The handle returned from a previous [WinCreateAtomTable](#) or [WinQuerySystemAtomTable](#) function.

WinQueryAtomUsage Parameter - atom

atom ([ATOM](#)) - input
Atom whose use count is to be returned.

WinQueryAtomUsage Return Value - ulcount

ulcount ([ULONG](#)) - returns
Use count of the atom.

65535	Integer atom
0	The specified atom or the atom table is invalid
Other	Use count.

WinQueryAtomUsage - Parameters

hatomtblAtomTbl ([HATOMTBL](#)) - input
Atom-table handle.

The handle returned from a previous [WinCreateAtomTable](#) or [WinQuerySystemAtomTable](#) function.

atom ([ATOM](#)) - input
Atom whose use count is to be returned.

ulcount ([ULONG](#)) - returns
Use count of the atom.

65535	Integer atom
0	The specified atom or the atom table is invalid
Other	Use count.

WinQueryAtomUsage - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HATOMTBL (0x1013)
An invalid atom-table handle was specified.

PMERR_INVALID_ATOM (0x1014)
The specified atom does not exist in the atom table.

WinQueryAtomUsage - Related Functions

Related Functions

- [WinAddAtom](#)
- [WinCreateAtomTable](#)
- [WinDeleteAtom](#)
- [WinDestroyAtomTable](#)
- [WinFindAtom](#)
- [WinQueryAtomLength](#)
- [WinQueryAtomName](#)
- [WinQueryAtomUsage](#)
- [WinQuerySystemAtomTable](#)

WinQueryAtomUsage - Example Code

This function returns the number of times an atom has been used.

```
#define INCL_WINATOM
#include <OS2.H>

HATOMTBL atomtbl;
ATOM atom = 25;

WinQueryAtomlength(atomtbl,
    atom);
```

WinQueryAtomUsage - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinQueryButtonCheckstate

WinQueryButtonCheckstate - Syntax

This macro returns the checked state of the button control specified.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND     hwndDlg;      /* Dialog window handle. */
USHORT    usId;         /* Button control identity. */
USHORT    usRetChkst;   /* Returns the checkstate of the specified button control. */

usRetChkst = WinQueryButtonCheckstate(hwndDlg,
                                       usId);
```

WinQueryButtonCheckstate Parameter - hwndDlg

hwndDlg ([HWND](#)) - input
Dialog window handle.

WinQueryButtonCheckstate Parameter - usId

usId ([USHORT](#)) - input
Button control identity.

WinQueryButtonCheckstate Return Value - usRetChkst

usRetChkst ([USHORT](#)) - returns
Returns the checkstate of the specified button control.

WinQueryButtonCheckstate - Parameters

hwndDlg ([HWND](#)) - input
Dialog window handle.

usId ([USHORT](#)) - input
Button control identity.

usRetChkst ([USHORT](#)) - returns
Returns the checkstate of the specified button control.

WinQueryButtonCheckstate - Remarks

This macro expands to:

```
#define WinQueryButtonCheckstate(hwndDlg, usId)
( (USHORT)WinSendDlgItemMsg(hwndDlg,
                             usId,
                             BM_QUERYCHECK,
                             (MPARAM)NULL,
                             (MPARAM)NULL) )
```

This function requires the existence of a message queue.

WinQueryButtonCheckstate - Related Functions

Related Functions

- [WinSendDlgItemMsg](#)
-

WinQueryButtonCheckstate - Related Messages

Related Messages

- [BM_QUERYCHECK](#)

WinQueryButtonCheckstate - Example Code

This function returns the checked state of the button control specified.

```
#define INCL_WINWINDOWMGR
#define INCL_WINBUTTONS
#include <OS2.H>

#define IDM_BUTTONA 900

HWND hwndDlg;
USHORT ChkState;

ChkState = WinQueryButtonCheckstate(hwndDlg,
                                     IDM_BUTTONA);

switch (ChkState)
{

    case 0:

        /* Unchecked */
        break;
    case 1:

        /* Checked */
        break;
    case 2:

        /* Indeterminate. */
        break;
}
```

WinQueryButtonCheckstate - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Example Code](#)
- [Related Functions](#)
- [Related Messages](#)
- [Glossary](#)

WinQueryCapture

WinQueryCapture - Syntax

This function returns the handle of the window that has the pointer captured.

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndDesktop; /* Desktop-window handle. */
HWND    hwnd;         /* Handle of the window with the pointer captured. */

hwnd = WinQueryCapture(hwndDesktop);
```

WinQueryCapture Parameter - hwndDesktop

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

WinQueryCapture Return Value - hwnd

hwnd ([HWND](#)) - returns
Handle of the window with the pointer captured.

NULLHANDLE	No window has the pointer captured, or an error occurred
Handle	Handle of the window with the pointer captured.

WinQueryCapture - Parameters

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

hwnd ([HWND](#)) - returns
Handle of the window with the pointer captured.

NULLHANDLE	No window has the pointer captured, or an error occurred
Handle	Handle of the window with the pointer captured.

WinQueryCapture - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinQueryCapture - Related Functions

Related Functions

- [WinQueryCapture](#)
 - [WinSetCapture](#)
-

WinQueryCapture - Example Code

This function returns the handle of the window that has the pointer captured.

```
#define INCL_WININPUT
#define INCL_WINWINDOWMGR
#include <OS2.H>
HWND hwnd; /* handle of window that has pointer captured */

hwnd = WinQueryCapture(HWND_DESKTOP); /* window that has */
/* pointer captured */
```

WinQueryCapture - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryClassInfo

WinQueryClassInfo - Syntax

This function returns window class information.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;           /* Anchor-block handle. */
PSZ      PSZClassName;  /* Class name. */
PCLASSINFO PclsClassInfo; /* Class information structure. */
BOOL     rc;           /* Class-exists indicator. */

rc = WinQueryClassInfo(hab, PSZClassName,
                       PclsClassInfo);
```

WinQueryClassInfo Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinQueryClassInfo Parameter - PSZClassName

PSZClassName (**PSZ**) - input
Class name.

WinQueryClassInfo Parameter - PclsClassInfo

PclsClassInfo (**PCLASSINFO**) - output
Class information structure.

WinQueryClassInfo Return Value - rc

rc (**BOOL**) - returns
Class-exists indicator.

TRUE
Class does exist

FALSE

Class does not exist.

WinQueryClassInfo - Parameters

hnb ([HAB](#)) - input
Anchor-block handle.

PSZClassName ([PSZ](#)) - input
Class name.

PclsiClassInfo ([PCLASSINFO](#)) - output
Class information structure.

rc ([BOOL](#)) - returns
Class-exists indicator.

TRUE

Class does exist

FALSE

Class does not exist.

WinQueryClassInfo - Remarks

PSZClassName is either an application-specified name (as defined by the [WinRegisterClass](#) call) or the name of a preregistered WC_* class; Preregistered class names are of the form #nnnnn, where nnnnn is up to five digits corresponding to the value of the WC_* class name constant.

This function provides information that is needed to create a subclass of a given class (see [WinSubclassWindow](#)).

WinQueryClassInfo - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_INTEGER_ATOM (0x1016)
The specified atom is not a valid integer atom.

PMERR_INVALID_ATOM_NAME (0x1015)
An invalid atom name string was passed.

PMERR_ATOM_NAME_NOT_FOUND (0x1017)
The specified atom name is not in the atom table.

WinQueryClassInfo - Related Functions

Related Functions

- [WinCalcFrameRect](#)

- [WinCreateFrameControls](#)
- [WinCreateStdWindow](#)
- [WinCreateWindow](#)
- [WinDefWindowProc](#)
- [WinDestroyWindow](#)
- [WinQueryClassInfo](#)
- [WinQueryClassName](#)
- [WinRegisterClass](#)
- [WinSubclassWindow](#)

WinQueryClassInfo - Example Code

This example obtains a pointer to the window procedure of the window class WC_COMBOBOX.

```
#define INCL_WINWINDOWMGR
#define INCL_WINENTRYFIELDS

#include <OS2.H>

HAB      hab;
CLASSINFO classinfo;
PFNWP    pWindowProc;

WinQueryClassInfo(hab,
                  WC_COMBOBOX,
                  &classinfo);

pWindowProc = classinfo.pfnWindowProc;
```

WinQueryClassInfo - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryClassName

WinQueryClassName - Syntax

This function copies the window class name, as a null-terminated string, into a buffer.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd;        /* Window handle. */
LONG    lLength;      /* Length of PCHBuffer. */
PCH      PCHBuffer;    /* Class name. */
LONG     lRetLen;      /* Returned class name length. */

lRetLen = WinQueryClassName(hwnd, lLength,
                             PCHBuffer);
```

WinQueryClassName Parameter - hwnd

hwnd (**HWND**) - input
Window handle.

If this window is of any of the preregistered WC_* classes the class name returned in the *PCHBuffer* parameter is in the form "#nnnnn", where "nnnnn" is a group of up to five digits that corresponds to the value of the WC_* class name constant.

WinQueryClassName Parameter - lLength

lLength (**LONG**) - input
Length of *PCHBuffer*.

It must be greater than 0.

WinQueryClassName Parameter - PCHBuffer

PCHBuffer (**PCH**) - output
Class name.

If the class name is longer than (*lLength*-1) only the first (*lLength*-1) characters of class name are copied.

WinQueryClassName Return Value - lRetLen

lRetLen (**LONG**) - returns
Returned class name length.

This is the length, *excluding* the null-termination character.

WinQueryClassName - Parameters

hwnd ([HWND](#)) - input
Window handle.

If this window is of any of the preregistered `WC_*` classes the class name returned in the *PCHBuffer* parameter is in the form `"#nnnnn"`, where `"nnnnn"` is a group of up to five digits that corresponds to the value of the `WC_*` class name constant.

Length ([LONG](#)) - input
Length of *PCHBuffer*.

It must be greater than 0.

PCHBuffer ([PCH](#)) - output
Class name.

If the class name is longer than $(Length-1)$ only the first $(Length-1)$ characters of class name are copied.

IRetLen ([LONG](#)) - returns
Returned class name length.

This is the length, *excluding* the null-termination character.

WinQueryClassName - Errors

Possible returns from [WinGetLastError](#)

`PMERR_INVALID_HWND (0x1001)`
An invalid window handle was specified.

`PMERR_INVALID_STRING_PARM (0x100B)`
The specified string parameter is invalid.

WinQueryClassName - Related Functions

Related Functions

- [WinCalcFrameRect](#)
 - [WinCreateFrameControls](#)
 - [WinCreateStdWindow](#)
 - [WinCreateWindow](#)
 - [WinDefWindowProc](#)
 - [WinDestroyWindow](#)
 - [WinQueryClassInfo](#)
 - [WinQueryClassName](#)
 - [WinRegisterClass](#)
 - [WinSubclassWindow](#)
-

WinQueryClassName - Example Code

This example obtains a pointer to the window procedure of the window class, given that we know the window handle.

```
#define INCL_WINWINDOWMGR

#include <OS2.H>

HAB      hab;
HWND     hwnd;
CLASSINFO classinfo;
PFNWP    pWindowProc;
char      *classname;

WinQueryClassName(hwnd,
                  sizeof(classname),
                  classname);

WinQueryClassInfo(hwnd,
                  classname,
                  &classinfo);

pWindowProc = classinfo.pfnWindowProc;
```

WinQueryClassName - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryClassThunkProc

WinQueryClassThunkProc - Syntax

This call queries the pointer-conversion procedure associated with a class.

```
#define INCL_WINTHUNKAPI /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

PSZ      pszClassName; /* Window-class name. */
PFN      thunkpr;      /* Pointer-conversion procedure identifier. */

thunkpr = WinQueryClassThunkProc(pszClassName);
```

WinQueryClassThunkProc Parameter - pszClassName

pszClassName ([PSZ](#)) - input
Window-class name.

WinQueryClassThunkProc Return Value - thunkpr

thunkpr ([PFN](#)) - returns
Pointer-conversion procedure identifier.

NULL	No pointer-conversion procedure is associated with this class.
Other	Identifier of the pointer-conversion procedure associated with this class.

WinQueryClassThunkProc - Parameters

pszClassName ([PSZ](#)) - input
Window-class name.

thunkpr ([PFN](#)) - returns
Pointer-conversion procedure identifier.

NULL	No pointer-conversion procedure is associated with this class.
Other	Identifier of the pointer-conversion procedure associated with this class.

WinQueryClassThunkProc - Related Functions

Related Functions

- [WinQueryClassThunkProc](#)
- [WinQueryWindowModel](#)
- [WinQueryWindowThunkProc](#)
- [WinSetClassThunkProc](#)
- [WinSetWindowThunkProc](#)

WinQueryClassThunkProc - Example Code

This example obtains the pointer conversion procedure of the window class, given that we have an anchor-block handle.


```

#define INCL_WINWINDOWMGR
#define INCL_WINTHUNKAPI
#include <OS2.H>
HWND hwnd;
/* . */
PFN pfn;
char *classname;

WinQueryClassName(hwnd,
                  sizeof(classname),
                  classname);

pfn = WinQueryClassThunkProc(classname);

```

WinQueryClassThunkProc - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryClipbrdData

WinQueryClipbrdData - Syntax

This function obtains a handle to the current clipboard data with a specified format.

```

#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;      /* Anchor-block handle. */
ULONG    fmt;      /* Format of the data to be accessed. */
ULONG    ulRet;    /* Handle to the clipboard data. */

ulRet = WinQueryClipbrdData(hab, fmt);

```

WinQueryClipbrdData Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinQueryClipbrdData Parameter - fmt

fmt ([ULONG](#)) - input
Format of the data to be accessed.

In addition to the predefined formats, private formats can be created using the standard system atom manager. The following is the list of standard clipboard formats:

CF_TEXT	Text format. Each line ends with a carriage-return/line-feed combination. Tab characters separate fields within a line. A NULL character signals the end of the data.
CF_DSPTEXT	Text display format associated with private format.
CF_BITMAP	Bit map.
CF_DSPBITMAP	Bit-map display format associated with private format.
CF_METAFILE	Metafile.
CF_DSPMETAFILE	Metafile display format associated with private format.
CF_PALETTE	Palette.

WinQueryClipbrdData Return Value - ulRet

ulRet ([ULONG](#)) - returns
Handle to the clipboard data.

0	Format does not exist, or an error occurred
Other	Handle to the clipboard data.

WinQueryClipbrdData - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

fmt ([ULONG](#)) - input
Format of the data to be accessed.

In addition to the predefined formats, private formats can be created using the standard system atom manager. The following is the list of standard clipboard formats:

CF_TEXT	Text format. Each line ends with a carriage-return/line-feed combination. Tab characters separate fields within a line. A NULL character signals the end of the data.
CF_DSPTEXT	Text display format associated with private format.
CF_BITMAP	Bit map.
CF_DSPBITMAP	Bit-map display format associated with private format.
CF_METAFILE	Metafile.
CF_DSPMETAFILE	Metafile display format associated with private format.
CF_PALETTE	Palette.

ulRet ([ULONG](#)) - returns
Handle to the clipboard data.

0	Format does not exist, or an error occurred
Other	Handle to the clipboard data.

WinQueryClipbrdData - Remarks

The returned data handle must not be used after the [WinCloseClipbrd](#) function is called. For this reason, the application must either copy the data (if required for long-term use) or process the data before the [WinCloseClipbrd](#) function is called. The application must neither free the data handle itself, nor leave it locked in any way.

Information about the format of the data in the clipboard can be obtained from [WinQueryClipbrdFmtInfo](#).

WinQueryClipbrdData - Related Functions

Related Functions

- [WinCloseClipbrd](#)
- [WinEmptyClipbrd](#)
- [WinEnumClipbrdFmts](#)
- [WinOpenClipbrd](#)
- [WinQueryClipbrdData](#)
- [WinQueryClipbrdFmtInfo](#)
- [WinQueryClipbrdOwner](#)
- [WinQueryClipbrdViewer](#)
- [WinSetClipbrdData](#)
- [WinSetClipbrdOwner](#)
- [WinSetClipbrdViewer](#)

WinQueryClipbrdData - Example Code

This example obtains a handle to the current clipboard data of text format.

```
#define INCL_WINCLIPBOARD
#include <OS2.H>
HAB hab;
/* . */
ULONG hclipbrdData;

hclipbrdData = WinQueryClipbrdData(hab, CF_TEXT);
```

WinQueryClipbrdData - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryClipbrdFmtInfo

WinQueryClipbrdFmtInfo - Syntax

This function determines whether a particular format of data is present in the clipboard, and if so, provides information about that format.

```
#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
ULONG    fmt;          /* Format of the data to be queried. */
PULONG   prgfFmtInfo;  /* Memory model and usage flags. */
BOOL     rc;           /* Format-exists indicator. */

rc = WinQueryClipbrdFmtInfo(hab, fmt, prgfFmtInfo);
```

WinQueryClipbrdFmtInfo Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinQueryClipbrdFmtInfo Parameter - fmt

fmt ([ULONG](#)) - input

Format of the data to be queried.

In addition to the predefined formats, private formats can be created using the standard system atom manager. The following is the list of standard clipboard formats:

CF_BITMAP

Bit map.

CF_DSPBITMAP

Bit-map display format associated with private format.

CF_DSPMETAFILE

Metafile display format associated with private format.

CF_DSPTTEXT

Text display format associated with private format.

CF_METAFILE

Metafile.

CF_PALETTE

Palette.

CF_TEXT

Text format. Each line ends with a carriage-return/line-feed combination. Tab characters separate fields within a line. A NULL character signals the end of the data.

WinQueryClipbrdFmtInfo Parameter - prgfFmtInfo

prgfFmtInfo ([PULONG](#)) - output

Memory model and usage flags.

These are the usage flags set by the setting application; that is, the CFI_* flags of the *flFmtInfo* parameter of the [WinSetClipbrdData](#) function.

If the format is CF_BITMAP, CF_DSPBITMAP, CF_METAFILE or CF_DSPMETAFILE, *prgfFmtInfo* is set to CFI_HANDLE. If the format is CF_TEXT or CF_DSPTTEXT, *prgfFmtInfo* is set to CFI_POINTER. If the format is user-defined, *prgfFmtInfo* is set to the value used in the [WinSetClipbrdData](#) function.

WinQueryClipbrdFmtInfo Return Value - rc

rc ([BOOL](#)) - returns

Format-exists indicator.

TRUE

fmt exists in the clipboard and *prgfFmtInfo* is set

FALSE

fmt does not exist in the clipboard and *prgfFmtInfo* is not set.

hab (**HAB**) - input

Anchor-block handle.

fmt (ULONG) - input

Format of the data to be queried.

In addition to the predefined formats, private formats can be created using the standard system atom manager. The following is the list of standard clipboard formats:

CF BITMAP

Bit map.

CF DSPBITMAP

Bit-map display format associated with private format.

CF DSPMETAFILE

Metafile display format associated with private format.

CF_DSPTTEXT

Text display format associated with private format.

CF METAFILE

Metafile.

CF PALETTE

Palette.

CF TEXT

Text format. Each line ends with a carriage-return/line-feed combination. Tab characters separate fields within a line. A NULL character signals the end of the data.

prgfFmtInfo (**PULONG**) - output

Memory model and usage flags.

These are the usage flags set by the setting application; that is, the `CFL_*` flags of the `//FmtInfo` parameter of the `WinSetClipbrdData` function.

If the format is CF_BITMAP, CF_DSPBITMAP, CF_METAFILE or CF_DSPMETAFILE, *prgFmtInfo* is set to CFI_HANDLE. If the format is CF_TEXT or CF_DSPTXT, *prgFmtInfo* is set to CFI_POINTER. If the format is user-defined, *prgFmtInfo* is set to the value used in the [WinSetClipbrdData](#) function.

rc (BOOL) - returns

Format-exists indicator.

TRUE

fmt exists in the clipboard and *prgfFmtInfo* is set

FALSE

fmt does not exist in the clipboard and *prgfFmtInfo* is not set.

This function does not cause the data to be rendered.

Applications can put information in the clipboard in their own private format. Use the following function calls to create and query a private format:

[illegible]

```
        "Private Data Type"),
    flFmtInfo);
```

```
ulData = WinQueryClipbrdData(hab,
                             WinFindAtom(WinQuerySystemAtomTable(),
                                           "Private Data Type"));
```

WinQueryClipbrdFmtInfo - Related Functions

Related Functions

- [WinCloseClipbrd](#)
- [WinEmptyClipbrd](#)
- [WinEnumClipbrdFmts](#)
- [WinOpenClipbrd](#)
- [WinQueryClipbrdData](#)
- [WinQueryClipbrdFmtInfo](#)
- [WinQueryClipbrdOwner](#)
- [WinQueryClipbrdViewer](#)
- [WinSetClipbrdData](#)
- [WinSetClipbrdOwner](#)
- [WinSetClipbrdViewer](#)

WinQueryClipbrdFmtInfo - Example Code

This example obtains a handle to the current clipboard data of text format if that format is present in the clipboard.

```
#define INCL_WINCLIPBOARD
#include <OS2.H>

HAB hab;

ULONG format;
ULONG hclipbrdData;

if (WinQueryClipbrdData(hab,CF_TEXT))
{
    hclipbrdData = WinQueryClipbrdFmfInfo(hab,
                                          CF_TEXT
                                          &format);
}
```

WinQueryClipbrdFmtInfo - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)

[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryClipbrdOwner

WinQueryClipbrdOwner - Syntax

This function obtains any current clipboard owner window.

```
#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;                /* Anchor-block handle. */
HWND     hwndClipbrdOwner;   /* Window handle of the current clipboard owner. */

hwndClipbrdOwner = WinQueryClipbrdOwner(hab);
```

WinQueryClipbrdOwner Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinQueryClipbrdOwner Return Value - hwndClipbrdOwner

hwndClipbrdOwner ([HWND](#)) - returns
Window handle of the current clipboard owner.

NULLHANDLE	If the clipboard is not owned by any window, or if an error occurred.
Other	Window handle of the current clipboard owner.

WinQueryClipbrdOwner - Parameters

hab ([HAB](#)) - input

Anchor-block handle.

hwndClipbrdOwner ([HWND](#)) - returns
Window handle of the current clipboard owner.

NULLHANDLE

If the clipboard is not owned by any window, or if an error occurred.

Other

Window handle of the current clipboard owner.

WinQueryClipbrdOwner - Related Functions

Related Functions

- [WinCloseClipbrd](#)
- [WinEmptyClipbrd](#)
- [WinEnumClipbrdFmts](#)
- [WinOpenClipbrd](#)
- [WinQueryClipbrdData](#)
- [WinQueryClipbrdFmtInfo](#)
- [WinQueryClipbrdOwner](#)
- [WinQueryClipbrdViewer](#)
- [WinSetClipbrdData](#)
- [WinSetClipbrdOwner](#)
- [WinSetClipbrdViewer](#)

WinQueryClipbrdOwner - Example Code

This example finds out which window currently owns the clipboard.

```
#define INCL_WINCLIPBOARD
#include <OS2.H>
HAB hab;
/* . */
HWND hwndClipbrdOwner;
```

```
hwndClipbrdOwner = WinQueryClipbrdOwner(hab);
```

WinQueryClipbrdOwner - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinQueryClipbrdViewer

WinQueryClipbrdViewer - Syntax

This function obtains any current clipboard viewer window.

```
#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
HWND     hwndClipbrdViewer; /* Current clipboard viewer window handle. */

hwndClipbrdViewer = WinQueryClipbrdViewer(
    hab);
```

WinQueryClipbrdViewer Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinQueryClipbrdViewer Return Value - hwndClipbrdViewer

hwndClipbrdViewer ([HWND](#)) - returns
Current clipboard viewer window handle.

NULLHANDLE	Clipboard does not have a current viewer window, or an error occurred
Other	Current clipboard viewer window handle.

WinQueryClipbrdViewer - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

hwndClipbrdViewer ([HWND](#)) - returns
Current clipboard viewer window handle.

NULLHANDLE	Clipboard does not have a current viewer window, or an error occurred
Other	Current clipboard viewer window handle.

WinQueryClipbrdViewer - Related Functions

Related Functions

- [WinCloseClipbrd](#)
- [WinEmptyClipbrd](#)
- [WinEnumClipbrdFmts](#)
- [WinOpenClipbrd](#)
- [WinQueryClipbrdData](#)
- [WinQueryClipbrdFmtInfo](#)
- [WinQueryClipbrdOwner](#)
- [WinQueryClipbrdViewer](#)
- [WinSetClipbrdData](#)
- [WinSetClipbrdOwner](#)
- [WinSetClipbrdViewer](#)

WinQueryClipbrdViewer - Example Code

This example finds out which window currently owns the clipboard.

```
#define INCL_WINCLIPBOARD
#include <OS2.H>
HAB hab;
/* . */
HWND hwndClipbrdViewer;
```

```
hwndClipbrdViewer = WinQueryClipbrdViewer(hab);
```

WinQueryClipbrdViewer - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryControlColors

WinQueryControlColors - Syntax

This function queries the colors currently being used by a control window.

Note: If you are interested in the default colors used by PM controls, see [Colors Used by PM Controls](#).

```
#define INCL_WINSYS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwnd;      /* Window handle. */
LONG      clrType;    /* Control color index. */
ULONG     fCtlColor;  /* Control Color Flags. */
ULONG     cCtlColor;  /* Number of items available in pCtlColor. */
PCTL_COLOR pCtlColor; /* Array of control colors, each comprising an index and value pair. */
LONG      rc;         /* Number of control colors returned in pCtlColor. */

rc = WinQueryControlColors(hwnd, clrType,
                           fCtlColor, cCtlColor, pCtlColor);
```

WinQueryControlColors Parameter - hwnd

hwnd (**HWND**) - input
Window handle.

WinQueryControlColors Parameter - clrType

clrType (**LONG**) - input
Control color index.

Must be one of the following control color indexes:.

CCT_STATIC	Static bitmap.
CCT_STATICTEXT	Static text.
CCT_GROUPBOX	Group Box.
CCT_PUSHBUTTON	Push button.
CCT_CHECKBOX	Check box.
CCT_RADIOBUTTON	Radio button.
CCT_ENTRYFIELD	Entry field.
CCT_LISTBOX	List box.
CCT_COMBOBOX	Combo box.
CCT_SCROLLBAR	Scroll bar.
CCT_FRAME	Window frame.
CCT_MENU	Menu.

CCT_TITLEBAR	Title bar.
CCT_SPINBUTTON	Spin button.
CCT_SLIDER	Slider.
CCT_CIRCULARSLIDER	Circular slider.
CCT_VALUESET	Value set.
CCT_MLE	Multiple line entry field.
CCT_CONTAINER	Container.
CCT_NOTEBOOK	Notebook.

WinQueryControlColors Parameter - fCtlColor

fCtlColor ([ULONG](#)) - input
Control Color Flags.

CCF_GLOBAL	Query global default colors (default)
CCF_APPLICATION	Query application default colors (used on current thread)
CCF_COUNTCOLORS	Return the number of color indexes used by a control window
CCF_ALLCOLORS	Return information on all colors used by a control window

WinQueryControlColors Parameter - cCtlColor

cCtlColor ([ULONG](#)) - input
Number of items available in *pCtlColor*.

If CCF_ALLCOLORS is specified, this value is the maximum number of items that can be returned in *pCtlColor*. Otherwise, this value is the number of color indexes being queried in *pCtlColor*.

If CCF_COUNTCOLORS is specified, this value is 0.

WinQueryControlColors Parameter - pCtlColor

pCtlColor ([PCTLCOLOR](#)) - in/out
Array of control colors, each comprising an index and value pair.

WinQueryControlColors Return Value - rc

rc (**LONG**) - returns
Number of control colors returned in *pCtlColor*.

TRUE
FALSE

Number of control colors returned.
Error, invalid parameters.

WinQueryControlColors - Parameters

hwnd (**HWND**) - input
Window handle.

clrType (**LONG**) - input
Control color index.

Must be one of the following control color indexes:.

CCT_STATIC
Static bitmap.

CCT_STATICTEXT
Static text.

CCT_GROUPBOX
Group Box.

CCT_PUSHBUTTON
Push button.

CCT_CHECKBOX
Check box.

CCT_RADIOBUTTON
Radio button.

CCT_ENTRYFIELD
Entry field.

CCT_LISTBOX
List box.

CCT_COMBOBOX
Combo box.

CCT_SCROLLBAR
Scroll bar.

CCT_FRAME
Window frame.

CCT_MENU
Menu.

CCT_TITLEBAR
Title bar.

CCT_SPINBUTTON
Spin button.

CCT_SLIDER
Slider.

CCT_CIRCULARSLIDER
Circular slider.

CCT_VALUESET
Value set.

CCT_MLE
Multiple line entry field.

CCT_CONTAINER
Container.

CCT_NOTEBOOK
Notebook.

fCtlColor (**ULONG**) - input
Control Color Flags.

CCF_GLOBAL
Query global default colors (default)

CCF_APPLICATION
Query application default colors (used on current thread)

CCF_COUNTCOLORS
Return the number of color indexes used by a control window

CCF_ALLCOLORS
Return information on all colors used by a control window

cCtlColor ([ULONG](#)) - input

Number of items available in *pCtlColor*.

If CCF_ALLCOLORS is specified, this value is the maximum number of items that can be returned in *pCtlColor*. Otherwise, this value is the number of color indexes being queried in *pCtlColor*.

If CCF_COUNTCOLORS is specified, this value is 0.

pCtlColor ([PCTLCOLOR](#)) - in/out

Array of control colors, each comprising an index and value pair.

rc ([LONG](#)) - returns

Number of control colors returned in *pCtlColor*.

TRUE

Number of control colors returned.

FALSE

Error, invalid parameters.

WinQueryControlColors - Remarks

This function returns the colors (one or more) that are currently being used by a control window of the type specified by *clrType*. If *hwnd* is a valid control window handle, the colors being used by that window are returned, including any presentation parameters it might be using. If *hwnd* is set to HWND_DESKTOP, the default colors being used by that window are returned.

To find out the default control colors for the current thread, specify CCF_APPLICATION. Be sure to issue the call to WinQueryColors from the same thread that created the control windows, otherwise the information returned will be incorrect.

To get all the colors used by a control window returned at once, specify CCF_ALLCOLORS. You can call WinQueryColors with CCF_COUNTCOLORS first to determine the buffer size you will need for the color information.

Each item in the returned array contains an index and value pair (CTL_COLOR). If CCF_ALLCOLORS and CCF_COUNTCOLORS are not set, each item's index should be set to a valid CCI_* constant for which a value is to be returned. The values returned in the array are always valid RGB colors (including SYSCLR_* values, which are actually indexes but can be used by graphics functions in RGB mode).

If an invalid CCI_* index is specified, CCF_NOTFOUND is returned.

Note: This function can only be used for querying the colors of standard PM controls.

WinQueryControlColors - Related Functions

Related Functions

- [WinSetControlColors](#)

WinQueryControlColors - Example Code

This example first queries the count of colors used by a push button. Then it queries all the colors used by push buttons in the application.

```
/* ***** */
/* Query the number of colors used by a push button. */
/* ***** */

cCount = WinQueryControlColors(
    HWND_DESKTOP,          /* Desktop window handle */
    CCT_PUSHBUTTON,        /* Select push button */
    CCF_COUNTCOLORS,       /* Count number of colors */
    0, 0);

pactlColor = (PCTLCOLOR)malloc(sizeof(CTLCOLOR) * cCount);

/* ***** */
/* Query all the colors used by push buttons in application. */
/* ***** */

WinQueryControlColors(HWND_DESKTOP,          /* Desktop window handle */
    CCT_PUSHBUTTON,        /* Select push button */
    CCF_ALLCOLORS |        /* Return all colors ... */
    CCF_APPLICATION,       /* ... for application */
    cCount,                /* Size of array */
    pactlColor);           /* Buffer for color data */
```

WinQueryControlColors - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryCp

WinQueryCp - Syntax

This function returns the queue code page for the specified message queue.

```
#define INCL_WINCOUNTRY /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HMQ      hmq;          /* Message queue. */
ULONG    ulCodePage;    /* Code page. */

ulCodePage = WinQueryCp(hmq);
```

WinQueryCp Parameter - hmq

hmq ([HMQ](#)) - input
Message queue.

WinQueryCp Return Value - ulCodePage

ulCodePage ([ULONG](#)) - returns
Code page.

0	Error occurred
Other	Queue code page for the specified message queue.

WinQueryCp - Parameters

hmq ([HMQ](#)) - input
Message queue.

ulCodePage ([ULONG](#)) - returns
Code page.

0	Error occurred
Other	Queue code page for the specified message queue.

WinQueryCp - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HMQ (0x1002)
An invalid message-queue handle was specified.

WinQueryCp - Related Functions

Related Functions

- [WinCpTranslateChar](#)
- [WinCpTranslateString](#)
- [WinQueryCp](#)
- [WinQueryCpList](#)
- [WinSetCp](#)

WinQueryCp - Example Code

This example returns the queue code page for the specified queue.

```
#define INCL_WINCOUNTRY
#include <OS2.H>

HMQ hmq;
/* . */
ULONG cp;

cp = WinQueryCp(hmq);
```

WinQueryCp - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryCpList

WinQueryCpList - Syntax

This function queries available code pages.

```
#define INCL_WINCOUNTRY /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
ULONG    ulcount;      /* Maximum number of code pages returned. */
PULONG   aulCodepage;  /* Code page list. */
ULONG    ulTotCount;   /* Total number of code pages available. */

ulTotCount = WinQueryCpList(hab, ulcount,
                           aulCodepage);
```

WinQueryCpList Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinQueryCpList Parameter - ulcount

ulcount ([ULONG](#)) - input
Maximum number of code pages returned.

WinQueryCpList Parameter - aulCodepage

aulCodepage ([PULONG](#)) - output
Code page list.

An array of *ulcount* elements, that contains a list of code pages available to the program.

For more information about code pages, see

WinQueryCpList Return Value - ulTotCount

ulTotCount ([ULONG](#)) - returns
Total number of code pages available.

0	An error occurred
Other	Total number of code pages available.

WinQueryCpList - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

ulcount ([ULONG](#)) - input
Maximum number of code pages returned.

aulCodepage ([PULONG](#)) - output
Code page list.

An array of *ulcount* elements, that contains a list of code pages available to the program.

For more information about code pages, see

ulTotCount ([ULONG](#)) - returns
Total number of code pages available.

0	An error occurred
Other	Total number of code pages available.

WinQueryCpList - Errors

Possible returns from [WinGetLastError](#)

PMERR_PARAMETER_OUT_OF_RANGE (0x1003)
The value of a parameter was not within the defined valid range for that parameter.

WinQueryCpList - Related Functions

- Related Functions**
- [WinCpTranslateChar](#)
 - [WinCpTranslateString](#)
 - [WinQueryCp](#)
 - [WinQueryCpList](#)
 - [WinSetCp](#)
-

WinQueryCpList - Example Code

This example queries available code pages.

```
#define INCL_WINCOUNTRY
#include <OS2.H>
#define maxcount 8
HAB hab;
/* . */
ULONG aulCodepage[maxcount];

WinQueryCpList(hab,
               (ULONG)maxcount,
               (PULONG) aulCodepage);
```

WinQueryCpList - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryCursorInfo

WinQueryCursorInfo - Syntax

This function obtains information about any current cursor.

```
#define INCL_WINCursors /* Or use INCL_WIN, INCL_PM, */  
#include <os2.h>  
  
HWND      hwndDesktop; /* Desktop-window handle. */  
PCURSORINFO pcsriCursorInfo; /* Cursor information. */  
BOOL      rc; /* Current-cursor indicator. */  
  
rc = WinQueryCursorInfo(hwndDesktop, pcsriCursorInfo);
```

WinQueryCursorInfo Parameter - hwndDesktop

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

WinQueryCursorInfo Parameter - pcsriCursorInfo

pcsriCursorInfo ([PCURSORINFO](#)) - output
Cursor information.

The values are equivalent to the parameters of the [WinCreateCursor](#) function except that *ulrgf* never includes the CURSOR_SETPOS option.

The size and position of the cursor are returned in window coordinates relative to the window identified by the parameter of the structure.

WinQueryCursorInfo Return Value - rc

rc ([BOOL](#)) - returns
Current-cursor indicator.

TRUE

Cursor exists

FALSE

Cursor does not exist, *pcsriCursorInfo* is not updated by this call.

WinQueryCursorInfo - Parameters

hwndDeskTop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP

The desktop-window handle

Other

Specified desktop-window handle.

pcsriCursorInfo ([PCCURSORINFO](#)) - output
Cursor information.

The values are equivalent to the parameters of the [WinCreateCursor](#) function except that *ulrgf* never includes the CURSOR_SETPOS option.

The size and position of the cursor are returned in window coordinates relative to the window identified by the parameter of the structure.

rc ([BOOL](#)) - returns
Current-cursor indicator.

TRUE

Cursor exists

FALSE

Cursor does not exist, *pcsriCursorInfo* is not updated by this call.

WinQueryCursorInfo - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinQueryCursorInfo - Related Functions

Related Functions

- [WinCreateCursor](#)
 - [WinDestroyCursor](#)
 - [WinQueryCursorInfo](#)
 - [WinShowCursor](#)
-

WinQueryCursorInfo - Example Code

This example obtains information about any current cursor.

```
#define INCL_WINCURSORS
#define INCL_WINWINDOWMGR
#include <OS2.H>
HWND hwnd; /* handle of window that has pointer captured */
CURSORINFO cursorinfo;

WinQueryCursorInfo(hwnd_DESKTOP, /* get cursor info */
                   &cursorinfo);
```

WinQueryCursorInfo - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryDesktopBkgnd

WinQueryDesktopBkgnd - Syntax

This function returns the desktop structure, which contains the information about the current state of the desktop background.

```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndDesktop; /* Desktop-window handle. */
PDESKTOP  pdsk;        /* Desktop-state structure. */
BOOL      rc;          /* Success indicator. */

rc = WinQueryDesktopBkgnd(hwndDesktop, pdsk);

```

WinQueryDesktopBkgnd Parameter - hwndDesktop

hwndDesktop (**HWND**) - input
Desktop-window handle.

HWND_DESKTOP	The desktop window
Other	Specified desktop window.

WinQueryDesktopBkgnd Parameter - pdsk

pdsk (**PDESKTOP**) - output
Desktop-state structure.

WinQueryDesktopBkgnd Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Desktop-window status provided
FALSE	Desktop-window status not provided.

WinQueryDesktopBkgnd - Parameters

hwndDesktop (**HWND**) - input
Desktop-window handle.

HWND_DESKTOP	The desktop window
Other	Specified desktop window.

pdesk ([PDESKTOP](#)) - output
Desktop-state structure.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Desktop-window status provided
FALSE	Desktop-window status not provided.

WinQueryDesktopBkgnd - Remarks

This function allows an application to query the background information of the desktop window. This application must be acting as the OS/2 PM shell in place of the IBM supplied shell. If the IBM supplied shell is executing it maintains control of the background of the desktop window, and WinQueryDesktopBkgnd will have no effect on the desktop window background, but will indicate a successful return code.

WinQueryDesktopBkgnd - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinQueryDesktopBkgnd - Related Functions

Related Functions

- [WinSetDesktopBkgnd](#)

WinQueryDesktopBkgnd - Example Code

This example uses WinQueryDesktopBkgnd to query the current desktop background bit map before setting it to a new bit map with WinSetDesktopBkgnd.

```
#define INCL_WINDESKTOP
#define INCL_WINWINDOWMGR
#include <OS2.H>
HAB hab;
HWND hwndDeskTop;
DESKTOP DeskTopState;
HBITMAP hbm;
HBITMAP hbm_user;

WinQueryDesktopBkgnd(HWND_DESKTOP,
                    &DeskTopState);

if (hbm_user != DeskTopState.hbm)
```

```

{
    DeskTopState.fl = SDT_LOADFILE;
    /* the szFile is used to load the bit map because */
    /* the fl parameter is set to SDT_LOADFILE.      */
    strcpy(DeskTopState.szFile, "fruit.bmp");
    DeskTopState.hbm = hbm_user;
    WinSetDesktopBkgnd(hwndDeskTop,
                      &DeskTopState);
}

```

WinQueryDesktopBkgnd - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryDesktopWindow

WinQueryDesktopWindow - Syntax

This function returns the desktop-window handle.

```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;           /* Anchor-block handle. */
HDC      hdc;           /* Device-context handle. */
HWND     hwndDeskTop;   /* Desktop-window handle. */

hwndDeskTop = WinQueryDesktopWindow(hab, hdc);

```

WinQueryDesktopWindow Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinQueryDesktopWindow Parameter - hdc

hdc ([HDC](#)) - input
Device-context handle.

NULLHANDLE
Default device (the screen).

WinQueryDesktopWindow Return Value - hwndDeskTop

hwndDeskTop ([HWND](#)) - returns
Desktop-window handle.

NULLHANDLE
Error occurred

Other
Desktop-window handle.

WinQueryDesktopWindow - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

hdc ([HDC](#)) - input
Device-context handle.

NULLHANDLE
Default device (the screen).

hwndDeskTop ([HWND](#)) - returns
Desktop-window handle.

NULLHANDLE
Error occurred

Other
Desktop-window handle.

WinQueryDesktopWindow - Remarks

Only the screen device supports windowing.

Many of the calls that require a desktop-window handle accept HWND_DESKTOP instead. For example, [WinCreateWindow](#) accepts HWND_DESKTOP for the parent-window handle to create a main window that is a child of the desktop window.

WinQueryDesktopWindow - Errors

Possible returns from [WinGetLastError](#)

PMERR_INV_HDC (0x207C)

An invalid device-context handle or (micro presentation space) presentation-space handle was specified.

WinQueryDesktopWindow - Related Functions

Related Functions

- [WinEnableWindow](#)
- [WinIsThreadActive](#)
- [WinIsWindow](#)
- [WinIsWindowEnabled](#)
- [WinQueryDesktopWindow](#)
- [WinQueryObjectWindow](#)
- [WinQueryWindowDC](#)
- [WinQueryWindowProcess](#)
- [WinQueryWindowRect](#)
- [WinWindowFromDC](#)
- [WinWindowFromID](#)
- [WinWindowFromPoint](#)

WinQueryDesktopWindow - Example Code

This function is used to find the desktop window handle. For most calls however, the parameter `HWND_DESKTOP` can be used.

```
#define INCL_WINDESKTOP
#include <OS2.H>
HAB hab;
HWND hwndDeskTop;

hwndDeskTop = WinQueryDesktopWindow(hab,
                                     NULLHANDLE);
```

WinQueryDesktopWindow - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryDlgItemShort

WinQueryDlgItemShort - Syntax

This function converts the text of a dialog item into an integer value.

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND     hwndDlg; /* Parent-window handle. */
ULONG     idItem; /* Identity of the child window whose text is to be converted. */
PSHORT    psResult; /* Integer value resulting from the conversion. */
BOOL      fSigned; /* Sign indicator. */
BOOL      rc; /* Success indicator. */

rc = WinQueryDlgItemShort(hwndDlg, idItem,
    psResult, fSigned);
```

WinQueryDlgItemShort Parameter - hwndDlg

hwndDlg (**HWND**) - input
Parent-window handle.

WinQueryDlgItemShort Parameter - idItem

idItem (**ULONG**) - input
Identity of the child window whose text is to be converted.

It must be greater or equal to 0 and less or equal to 0xFFFF.

WinQueryDlgItemShort Parameter - psResult

psResult (**PSHORT**) - output
Integer value resulting from the conversion.

WinQueryDlgItemShort Parameter - fSigned

fSigned (BOOL) - input
Sign indicator.

TRUE	Signed text. It is inspected for a minus sign (-).
FALSE	Unsigned text.

WinQueryDlgItemShort Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE	Successful conversion
FALSE	Error occurred.

WinQueryDlgItemShort - Parameters

hwndDlg (HWND) - input
Parent-window handle.

idItem (ULONG) - input
Identity of the child window whose text is to be converted.

It must be greater or equal to 0 and less or equal to 0xFFFF.

psResult (PSHORT) - output
Integer value resulting from the conversion.

fSigned (BOOL) - input
Sign indicator.

TRUE	Signed text. It is inspected for a minus sign (-).
FALSE	Unsigned text.

rc (BOOL) - returns
Success indicator.

TRUE	Successful conversion
FALSE	Error occurred.

WinQueryDlgItemShort - Remarks

This function is useful for converting a numerical input field into a binary number for further processing. The text of a dialog item is assumed to be an ASCII string.

This function is valid for any window with children. However, it is typically used for dialog items in a dialog window.

WinQueryDlgItemShort - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinQueryDlgItemShort - Related Functions

Related Functions

- [WinQueryDlgItemShort](#)
- [WinQueryDlgItemText](#)
- [WinQueryDlgItemTextLength](#)
- [WinQueryWindowText](#)
- [WinQueryWindowTextLength](#)
- [WinSetDlgItemShort](#)
- [WinSetDlgItemText](#)
- [WinSetWindowText](#)

WinQueryDlgItemShort - Example Code

This example gets the text from a Dialog Box entry field as an integer value.

```
#define INCL_WINDIALOGS

#include <OS2.H>

#define ID_ENTRYFLD 900

HAB hab;
HWND hwnd;
ULONG msg;
MPARAM mp1;
SHORT iconverted;

/* . */
switch(msg)
{
    case WM_INITDLG:

    case WM_COMMAND:
        switch(SHORT1FROMMP(mp1))
        {
            case DID_OK:
                WinQueryDlgItemShort(hwnd,
                                     ID_ENTRYFLD,
                                     &iconverted, /* integer result */
                                     TRUE); /* Get the short */
        }
    }
}
```

```
}  
}
```

WinQueryDlgItemShort - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinQueryDlgItemText

WinQueryDlgItemText - Syntax

This function queries a text string in a dialog item.

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMMON section */  
#include <os2.h>  
  
HWND    hwndDlg; /* Parent-window handle. */  
ULONG    idItem; /* Identity of the child window whose text is to be queried. */  
LONG     lMaxText; /* Length of pszText. */  
PSZ      pszText; /* Output string. */  
ULONG    ulRetLen; /* Actual number of characters returned. */  
  
ulRetLen = WinQueryDlgItemText(hwndDlg, idItem,  
                                lMaxText, pszText);
```

WinQueryDlgItemText Parameter - hwndDlg

hwndDlg ([HWND](#)) - input
Parent-window handle.

WinQueryDlgItemText Parameter - idItem

idItem ([ULONG](#)) - input
Identity of the child window whose text is to be queried.

It must be greater or equal to 0 and less or equal to 0xFFFF.

WinQueryDlgItemText Parameter - IMaxText

IMaxText ([LONG](#)) - input
Length of *pszText*.

It must be greater or equal to 0.

WinQueryDlgItemText Parameter - pszText

pszText ([PSZ](#)) - output
Output string.

This is the text string that is obtained from the dialog item.

WinQueryDlgItemText Return Value - ulRetLen

ulRetLen ([ULONG](#)) - returns
Actual number of characters returned.

0	Error occurred
Other	Actual number of characters returned, not including the null-terminating character. The maximum value is (<i>IMaxText</i> -1).

WinQueryDlgItemText - Parameters

hwndDlg ([HWND](#)) - input
Parent-window handle.

idItem ([ULONG](#)) - input
Identity of the child window whose text is to be queried.

It must be greater or equal to 0 and less or equal to 0xFFFF.

IMaxText ([LONG](#)) - input

Length of *pszText*.

It must be greater or equal to 0.

pszText ([PSZ](#)) - output
Output string.

This is the text string that is obtained from the dialog item.

ulRetLen ([ULONG](#)) - returns
Actual number of characters returned.

0

Error occurred

Other

Actual number of characters returned, not including the null-terminating character. The maximum value is (*lMaxText*-1).

WinQueryDlgItemText - Remarks

This function is valid for any window with children. However, it is typically used for dialog items in a dialog window.

WinQueryDlgItemText - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinQueryDlgItemText - Related Functions

Related Functions

- [WinQueryDlgItemShort](#)
 - [WinQueryDlgItemText](#)
 - [WinQueryDlgItemTextLength](#)
 - [WinQueryWindowText](#)
 - [WinQueryWindowTextLength](#)
 - [WinSetDlgItemShort](#)
 - [WinSetDlgItemText](#)
 - [WinSetWindowText](#)
-

WinQueryDlgItemText - Example Code

This example is the beginning of a function which processes the text which is displayed in the message text line.

```
#define INCL_WINDIALOGS
#include <OS2.H>
#define DID_MSGEDIT 900
```

```

void SelectMessageFromText(HWND hwndDlg)
{
    char    szTemp[80];

    /* First get the edit text from the string */
    WinQueryDlgItemText(hwndDlg, DID_MSGEDIT, sizeof(szTemp),
        (PSZ)szTemp);
    /* . */
    /* . */
}

```

WinQueryDlgItemText - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinQueryDlgItemTextLength

WinQueryDlgItemTextLength - Syntax

This function queries the length of the text string in a dialog item, not including any null termination character.

```

#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND    hwndDlg; /* Parent-window handle. */
ULONG   idItem; /* Identity of the child window whose text is to be queried. */
LONG     lRetLen; /* Length of text. */

lRetLen = WinQueryDlgItemTextLength(hwndDlg,
    idItem);

```

WinQueryDlgItemTextLength Parameter - hwndDlg

hwndDlg ([HWND](#)) - input
Parent-window handle.

WinQueryDlgItemTextLength Parameter - idItem

idItem ([ULONG](#)) - input
Identity of the child window whose text is to be queried.

It must be greater or equal to 0 and less or equal to 0xFFFF.

WinQueryDlgItemTextLength Return Value - IRetLen

IRetLen ([LONG](#)) - returns
Length of text.

0
Error occurred

Other
Length of text.

WinQueryDlgItemTextLength - Parameters

hwndDlg ([HWND](#)) - input
Parent-window handle.

idItem ([ULONG](#)) - input
Identity of the child window whose text is to be queried.

It must be greater or equal to 0 and less or equal to 0xFFFF.

IRetLen ([LONG](#)) - returns
Length of text.

0
Error occurred

Other
Length of text.

WinQueryDlgItemTextLength - Remarks

This function is valid for any window with children. However, it is typically used for dialog items in a dialog window.

WinQueryDlgItemTextLength - Related Functions

Related Functions

- [WinQueryDlgItemShort](#)
- [WinQueryDlgItemText](#)
- [WinQueryDlgItemTextLength](#)
- [WinQueryWindowText](#)
- [WinQueryWindowTextLength](#)
- [WinSetDlgItemShort](#)
- [WinSetDlgItemText](#)
- [WinSetWindowText](#)

WinQueryDlgItemTextLength - Example Code

This example is the beginning of a function which processes the text which is displayed in the message text line.

```
#define INCL_WINDIALOGS
#define INCL_DOSMEMMGR
#include <OS2.H>
#define DID_MSGEDIT 900
void SelectMessageFromText (hwndDlg)
HWND    hwndDlg;
{

char *szTemp;
LONG   length;

    /* First get the edit text from the string */

length = WinQueryDlgItemTextLength(hwndDlg,
                                   DID_MSGEDIT);
    /* now we know the buffer size needed. */

DosAllocMem( (PPVOID)szTemp,
             (ULONG)length,
             PAG_READ |
             PAG_WRITE |
             PAG_COMMIT);

WinQueryDlgItemText(hwndDlg,
                    DID_MSGEDIT,
                    sizeof(szTemp),
                    (PSZ)szTemp);

    /* . */
    /* . */
}
```

WinQueryDlgItemTextLength - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryFocus

WinQueryFocus - Syntax

This function returns the focus window. It is NULLHANDLE if there is no focus window.

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND     hwndDesktop; /* Desktop-window handle. */
HWND     hwndFocus;   /* Focus-handle. */

hwndFocus = WinQueryFocus(hwndDesktop);
```

WinQueryFocus Parameter - hwndDesktop

- hwndDesktop (HWND)** - input
Desktop-window handle.
 - HWND_DESKTOP
The desktop-window handle
 - Other
Specified desktop-window handle.

WinQueryFocus Return Value - hwndFocus

- hwndFocus (HWND)** - returns
Focus-handle.
 - NULL
Error occurred or no focus window.

WinQueryFocus - Parameters

- hwndDesktop (HWND)** - input
Desktop-window handle.
 - HWND_DESKTOP

	The desktop-window handle
Other	Specified desktop-window handle.
hwndFocus (HWND) - returns Focus-handle.	
NULL	Error occurred or no focus window.

WinQueryFocus - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinQueryFocus - Related Functions

Related Functions

- [WinEnablePhysInput](#)
- [WinFocusChange](#)
- [WinGetKeyState](#)
- [WinGetPhysKeyState](#)
- [WinQueryFocus](#)
- [WinSetFocus](#)
- [WinSetKeyboardStateTable](#)

WinQueryFocus - Example Code

This example checks to see if the menu has the focus.

```
#define INCL_WININPUT
#include <OS2.H>
#define SYS_MENU 900
HWND hwndFrame;

if (WinQueryFocus(HWND_DESKTOP) ==
    WinWindowFromID(hwndFrame, SYS_MENU))
{
    /* . */
}
```

WinQueryFocus - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryHelpInstance

WinQueryHelpInstance - Syntax

This function enables the application to query the instance of the Help Manager associated with the application-supplied window handle.

```
#define INCL_WINHELP /* Or use INCL_WIN, INCL_PM, */  
#include <os2.h>  
  
HWND    hwndApp;    /* Handle of the application window. */  
HWND    hwndHelp;    /* Help Manager window handle. */  
  
hwndHelp = WinQueryHelpInstance(hwndApp);
```

WinQueryHelpInstance Parameter - hwndApp

hwndApp ([HWND](#)) - input
Handle of the application window.

WinQueryHelpInstance Return Value - hwndHelp

hwndHelp ([HWND](#)) - returns
Help Manager window handle.

NULLHANDLE	No Help Manager instance is associated with the application window.
Other	Help Manager window handle.

WinQueryHelpInstance - Parameters

hwndApp ([HWND](#)) - input
Handle of the application window.

hwndHelp ([HWND](#)) - returns
Help Manager window handle.

NULLHANDLE
No Help Manager instance is associated with the application window.

Other
Help Manager window handle.

WinQueryHelpInstance - Remarks

The Help Manager first traces the parent window chain until it is NULLHANDLE or HWND_DESKTOP. Then Help Manager traces the owner chain. If a parent of the owner window exists, the trace begins again with the parent chain.

The window chain will be traced until the Help Manager finds an instance of the Help Manager or until both the parent and owner windows are NULLHANDLE or HWND_DESKTOP.

WinQueryHelpInstance - Related Functions

Related Functions

- [WinAssociateHelpInstance](#)
 - [WinCreateHelpInstance](#)
 - [WinCreateHelpTable](#)
 - [WinDestroyHelpInstance](#)
 - [WinLoadHelpTable](#)
 - [WinQueryHelpInstance](#)
-

WinQueryHelpInstance - Example Code

This example shows the use of the WinQueryHelpInstance call during the processing of a WM_INITMENU message in order to obtain the handle for sending an HM_SET_ACTIVE_WINDOW message.

```
#define INCL_WIN
#include <os2.h>

MRESULT wm_initmenu( HWND hWnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    /* Send message to establish the current window's parent      */
    /* as the active help window.                                  */
    WinSendMsg( WinQueryHelpInstance( hWnd ),
                HM_SET_ACTIVE_WINDOW,
                (MPARAM)WinQueryWindow( hWnd, QW_PARENT ),
                (MPARAM)WinQueryWindow( hWnd, QW_PARENT ) );

    /* Pass message on for default processing                      */
    return WinDefWindowProc( hWnd, ulMsg, mp1, mp2 );
}
```

WinQueryHelpInstance - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryLboxCount

WinQueryLboxCount - Syntax

This macro returns the number of items in the List Box.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND     hwndLbox; /* Listbox handle. */
LONG     lRetNumIt; /* Number of items in the list box. */

lRetNumIt = WinQueryLboxCount(hwndLbox);
```

WinQueryLboxCount Parameter - hwndLbox

hwndLbox ([HWND](#)) - input
Listbox handle.

WinQueryLboxCount Return Value - lRetNumIt

lRetNumIt ([LONG](#)) - returns
Number of items in the list box.

WinQueryLboxCount - Parameters

hwndLbox ([HWND](#)) - input
Listbox handle.

lRetNumIt ([LONG](#)) - returns
Number of items in the list box.

WinQueryLboxCount - Remarks

This macro is defined as:

```
#define WinQueryLboxCount(hwndLbox) \
    ((LONG)WinSendMsg(hwndLbox, \
        LM_QUERYITEMCOUNT, \
        (MPARAM)NULL, \
        (MPARAM)NULL))
```

This macro requires the existence of a message queue.

WinQueryLboxCount - Related Functions

Related Functions

- [WinSendMsg](#)

WinQueryLboxCount - Related Messages

Related Messages

- [LM_QUERYITEMCOUNT](#)

WinQueryLboxCount - Example Code

This example uses WinQueryLboxCount to find the number of list box items and selects them all.

```
#define INCL_WINLISTBOXES
#define INCL_WINWINDOWMGR
#include <OS2.H>

LONG cWindows;
HWND hwndWindowLB;
```

```

cWindows = WinQueryLboxCount(hwndWindowLB);

/* Loop through all windows, selecting them all */
while (cWindows)
{
    WinSendMsg(hwndWindowLB,
                LM_SELECTITEM,
                (MPARAM)--cWindows,
                (MPARAM)TRUE);
}

```

WinQueryLboxCount - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Related Messages](#)

[Glossary](#)

WinQueryLboxItemText

WinQueryLboxItemText - Syntax

This macro fills the buffer with the text of the indexed item. It returns the length of the text.

```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndLbox; /* List box handle. */
SHORT   index;    /* Index of the listbox item. */
PSZ     psz;      /* Pointer to a null terminated string. */
SHORT   cchMax;   /* Maximum number of characters allocated to the string. */
LONG    lRetTxtL; /* Actual text length copied. */

lRetTxtL = WinQueryLboxItemText(hwndLbox,
                                index, psz, cchMax);

```

WinQueryLboxItemText Parameter - hwndLbox

hwndLbox ([HWND](#)) - input
List box handle.

WinQueryLboxItemText Parameter - index

index ([SHORT](#)) - input
Index of the listbox item.

WinQueryLboxItemText Parameter - psz

psz ([PSZ](#)) - input
Pointer to a null terminated string.

WinQueryLboxItemText Parameter - cchMax

cchMax ([SHORT](#)) - input
Maximum number of characters allocated to the string.

WinQueryLboxItemText Return Value - IRetTxtL

IRetTxtL ([LONG](#)) - returns
Actual text length copied.

WinQueryLboxItemText - Parameters

hwndLbox ([HWND](#)) - input
List box handle.

index ([SHORT](#)) - input
Index of the listbox item.

psz ([PSZ](#)) - input
Pointer to a null terminated string.

cchMax ([SHORT](#)) - input
Maximum number of characters allocated to the string.

IRetTxtL ([LONG](#)) - returns
Actual text length copied.

WinQueryLboxItemText - Remarks

This macro is defined as:

```
#define WinQueryLboxItemText(hwndLbox, index, psz, cchMax) \
    ((LONG)WinSendMessage(hwndLbox, \
        LM_QUERYITEMTEXT, \
        MPFROM2SHORT((index), (cchMax)), \
        MPFROMP(psz)))
```

This macro requires the existence of a message queue.

WinQueryLboxItemText - Related Functions

Related Functions

- [WinSendMessage](#)
-

WinQueryLboxItemText - Related Messages

Related Messages

- [LM_INSERTITEM](#)
-

WinQueryLboxItemText - Example Code

This example uses WinQueryLboxItemText to copy all of the list box items into a buffer.

```
#define INCL_WINLISTBOXES
#define INCL_WINWINDOWMGR
#include <OS2.H>

LONG cWindows;
char *szTemp;
HWND hwndLB;
SHORT maxchar, index = 0;

cWindows = WinQueryLboxCount(hwndLB);

/* allocate a buffer for cWindows items. */

DosAllocMem((PPVOID)&szTemp,
    (ULONG)cWindows*256*sizeof(char),
```

```

        PAG_READ |
        PAG_WRITE |
        PAG_COMMIT);

/* loop through all of the items; copying each */
/* one the buffer. */

while (index <= cWindows)
{
    maxchar = WinQueryLboxItemTextLength(hwndLB,index);
    WinQueryLboxItemText(hwndLB,
                        index++,
                        szTemp,
                        maxchar);
    (*szTemp)+=maxchar*sizeof(char); /* increment pointer by number */
                                    /* of bytes copied. */
}

```

WinQueryLboxItemText - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Related Messages](#)

[Glossary](#)

WinQueryLboxItemTextLength

WinQueryLboxItemTextLength - Syntax

This macro returns the length of the text of the indexed item in the List Box.

```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndLbox; /* Listbox handle. */
SHORT   index;    /* Index of the item in the List Box. */
SHORT   sRetLen;  /* Text length of the indexed item. */

sRetLen = WinQueryLboxItemTextLength(hwndLbox,
                                     index);

```

WinQueryLboxItemTextLength Parameter - hwndLbox

hwndLbox ([HWND](#)) - input
Listbox handle.

WinQueryLboxItemTextLength Parameter - index

index ([SHORT](#)) - input
Index of the item in the List Box.

WinQueryLboxItemTextLength Return Value - sRetLen

sRetLen ([SHORT](#)) - returns
Text length of the indexed item.

WinQueryLboxItemTextLength - Parameters

hwndLbox ([HWND](#)) - input
Listbox handle.

index ([SHORT](#)) - input
Index of the item in the List Box.

sRetLen ([SHORT](#)) - returns
Text length of the indexed item.

WinQueryLboxItemTextLength - Remarks

This macro is defined as:

```
#define WinQueryLboxItemTextLength(hwndLbox, index) \
    ((SHORT)WinSendMessage(hwndLbox, \
        LM_QUERYITEMTEXTLENGTH, \
        MPFROMSHORT(index), \
        (MPARAM)NULL))
```

This macro requires the existence of a message queue.

WinQueryLboxItemTextLength - Related Functions

Related Functions

- [WinSendMsg](#)
-

WinQueryLboxItemTextLength - Related Messages

Related Messages

- [LM_QUERYITEMTEXTLENGTH](#)
-

WinQueryLboxItemTextLength - Example Code

This example uses WinQueryLboxItemText to copy all of the list box items into a buffer.

```
#define INCL_WINLISTBOXES
#define INCL_WINWINDOWMGR
#include <OS2.H>

LONG cWindows;
char *szTemp;
HWND hwndLB;
SHORT maxchar, index = 0;

cWindows = WinQueryLboxCount(hwndLB);

/* allocate a buffer for cWindows items. */

DosAllocMem((PVOID)&szTemp,
            (ULONG)cWindows*256*sizeof(char),
            PAG_READ |
            PAG_WRITE |
            PAG_COMMIT);

/* loop through all of the items; copying each */
/* one the buffer. */

while (index <= cWindows)
{
    maxchar = WinQueryLboxItemTextLength(hwndLB, index);
    WinQueryLboxItemText(hwndLB,
                        index++,
                        szTemp,
                        maxchar);
    (*szTemp)+=maxchar*sizeof(char); /* increment pointer by number */
                                   /* of bytes copied. */
}

}
```

WinQueryLboxItemTextLength - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)

[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinQueryLboxSelectedItem

WinQueryLboxSelectedItem - Syntax

This macro returns the index of the selected item in the List Box (for single selection only).

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndLbox; /* List box handle. */
LONG    lRetIndex; /* Index of the selected item. */

lRetIndex = WinQueryLboxSelectedItem(hwndLbox);
```

WinQueryLboxSelectedItem Parameter - hwndLbox

hwndLbox ([HWND](#)) - input
List box handle.

WinQueryLboxSelectedItem Return Value - lRetIndex

lRetIndex ([LONG](#)) - returns
Index of the selected item.

WinQueryLboxSelectedItem - Parameters

hwndLbox ([HWND](#)) - input
List box handle.

lRetIndex ([LONG](#)) - returns

Index of the selected item.

WinQueryLboxSelectedItem - Remarks

This macro is defined as:

```
#define WinQueryLBoxSelectedItem (hwndLbox)
    ((LONG)WinSendMsg(hwndLbox,
        LM_QUERYSELECTION,
        MPFROMLONG(LIT_FIRST),
        (MPARAM)NULL))
```

This macro requires the existence of a message queue.

WinQueryLboxSelectedItem - Related Functions

Related Functions

- [WinSendMsg](#)
-

WinQueryLboxSelectedItem - Related Messages

Related Messages

- [LM_QUERYSELECTION](#)
-

WinQueryLboxSelectedItem - Example Code

This example copies the text from the selected item in a list box to a buffer. Note that while WinQueryLboxSelectedItem returns a LONG value, WinQueryLboxItemText takes a SHORT parameter.

```
#define INCL_WINLISTBOXES
#define INCL_WINWINDOWMGR
#include <OS2.H>

HWND hwndLB;
LONG index;
char szTemp[256];

index = WinQueryLboxSelectedItem(hwndLB);

WinQueryLboxItemText(hwndLB,
    (SHORT) index,
    szTemp,
    WinQueryLboxItemTextLength(hwndLB, index));
```

WinQueryLboxSelectedItem - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinQueryMsgPos

WinQueryMsgPos - Syntax

This function returns the pointer position, in screen coordinates, when the last message obtained from the current message queue is posted.

```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, */  
#include <os2.h>  
  
HAB      hab; /* Anchor-block handle. */  
PPOINTL  pptl; /* Pointer position in screen coordinates. */  
BOOL     rc; /* Success indicator. */  
  
rc = WinQueryMsgPos(hab, pptl);
```

WinQueryMsgPos Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinQueryMsgPos Parameter - pptl

pptl ([PPOINTL](#)) - output
Pointer position in screen coordinates.

WinQueryMsgPos Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinQueryMsgPos - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

pptl ([PPOINTL](#)) - output
Pointer position in screen coordinates.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinQueryMsgPos - Remarks

The pointer position is the same as that in the parameter of a [QMSG](#) structure.

To obtain the current position of the pointer, use the [WinQueryPointerPos](#) function.

WinQueryMsgPos - Related Functions

Related Functions

- [WinBroadcastMsg](#)
- [WinCreateMsgQueue](#)
- [WinDestroyMsgQueue](#)
- [WinDispatchMsg](#)
- [WinGetDlgMsg](#)
- [WinGetMsg](#)
- [WinInSendMessage](#)
- [WinPeekMsg](#)
- [WinPostMsg](#)
- [WinPostQueueMsg](#)
- [WinQueryMsgPos](#)
- [WinQueryMsgTime](#)

- [WinQueryQueueInfo](#)
- [WinQueryQueueStatus](#)
- [WinSendDlgItemMsg](#)
- [WinSendMsg](#)
- [WinSetClassMsgInterest](#)
- [WinSetMsgInterest](#)
- [WinSetMsgMode](#)
- [WinSetSynchroMode](#)
- [WinWaitMsg](#)

WinQueryMsgPos - Example Code

This example returns position and time of the the last message obtained from the current message queue.

```
#define INCL_WINMESSAGEGR
#define INCL_WINDIALOGS
#include <OS2.H>
#include <stdio.h>
HAB hab;
POINTL ptl;
CHAR szMsg[100];
HWND hwnd;
ULONG ulTime;

WinQueryMsgPos(hab, &ptl);

ulTime = WinQueryMsgTime(hab);

sprintf(szMsg, "x = %ld   y = %ld\n\ntime = %ld",
        ptl.x, ptl.y, ulTime);
WinMessageBox(HWND_DESKTOP,
    hwnd,
    szMsg,
    "Debugging information",
    0,
    MB_NOICON | MB_OK);
```

WinQueryMsgPos - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryMsgTime

WinQueryMsgTime - Syntax

This function returns the message time for the last message retrieved by the [WinGetMsg](#) or [WinPeekMsg](#) functions from the current message queue.

```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;      /* Anchor-block handle. */
ULONG    ulTime;    /* Time in milliseconds. */

ulTime = WinQueryMsgTime(hab);
```

WinQueryMsgTime Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinQueryMsgTime Return Value - ulTime

ulTime ([ULONG](#)) - returns
Time in milliseconds.

WinQueryMsgTime - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

ulTime ([ULONG](#)) - returns
Time in milliseconds.

WinQueryMsgTime - Remarks

The message time is the time the message is posted, measured in milliseconds, from the time the system is started. Its value is the same as that in the parameter of the [QMSG](#) structure.

To calculate time delays between messages, the time of the first message is subtracted from the time of the second message.

Time values do not always increase because the value is the number of milliseconds since the system was started, and the system accumulator for this count can wrap through zero.

WinQueryMsgTime - Related Functions

Related Functions

- [WinGetCurrentTime](#)
 - [WinQueryMsgTime](#)
 - [WinStartTimer](#)
 - [WinStopTimer](#)
-

WinQueryMsgTime - Example Code

This example returns position and time of the the last message obtained from the current message queue.

```
#define INCL_WINMESSAGEGR
#define INCL_WINDIALOGS
#include <OS2.H>
#include <stdio.h>
HAB hab;
POINTL ptl;
CHAR szMsg[100];
HWND hwnd;
ULONG ulTime;

WinQueryMsgPos(hab, &ptl);

ulTime = WinQueryMsgTime(hab);

sprintf(szMsg, "x = %ld  y = %ld\n\ntime = %ld",
        ptl.x, ptl.y, ulTime);
WinMessageBox(HWND_DESKTOP,
        hwnd,
        szMsg,
        "Debugging information",
        0,
        MB_NOICON | MB_OK);
```

WinQueryMsgTime - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryObject

WinQueryObject - Syntax

The WinQueryObject function returns a handle to the given object.

```
#define INCL_WINWORKPLACE
#include <os2.h>

PSZ      pObjectID; /* The object ID of an existing object. */
HOBJECT  hObject;   /* MRESULT. */

hObject = WinQueryObject(pObjectID);
```

WinQueryObject Parameter - pObjectID

pObjectID (**PSZ**) - input

The object ID of an existing object.

The object ID of an existing object, for example <LOCATION_DESKTOP>, or alternatively the fully qualified filename of any file or directory.

WinQueryObject Return Value - hObject

hObject (**HOBJECT**) - returns
MRESULT.

Persistent object handle, or NULLHANDLE if the object does not exist or could not be awakened.

WinQueryObject - Parameters

pObjectID (**PSZ**) - input

The object ID of an existing object.

The object ID of an existing object, for example <LOCATION_DESKTOP>, or alternatively the fully qualified filename of any file or directory.

hObject (**HOBJECT**) - returns

MRESULT.

Persistent object handle, or NULLHANDLE if the object does not exist or could not be awakened.

WinQueryObject - Remarks

This function allows you to obtain the persistent object handle for any file object, by passing the fully qualified filename. Similarly any objects' handle can be retrieved if its object ID string is passed. Once a program has an object handle, it is able to change the objects state by using the [WinSetObjectData](#) function or delete the object using the [WinDestroyObject](#) function. Note that valid object ID strings must always start with the "<" character and be terminated by the ">" character, and are thus invalid file system names.

Using [HOBJECT](#) for .INI files or files in which an application uses a rename/save/delete sequence is not supported.

WinQueryObject - Related Functions

Related Functions

- [WinCreateObject](#)
 - [WinDestroyObject](#)
 - [WinSetObjectData](#)
-

WinQueryObject - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

WinQueryObjectPath

WinQueryObjectPath - Syntax

This function is specific to OS/2 Version 3.0 or higher.

This function returns the directory specification of a given object handle.

```
#define INCL_WINWORKPLACE
#include <os2.h>
```

```
HOBJECT    hObject;      /* Object handle of the object whose file/directory specification is to be returned.
PSZ         pszPathname; /* Memory allocated by caller in which directory specification is written. */
ULONG      ulSize;       /* Number of bytes pointed to by pszPathname. */
BOOL        rc;          /* Success indicator. */

rc = WinQueryObjectPath(hObject, pszPathname,
                        ulSize);
```

WinQueryObjectPath Parameter - hObject

hObject (**HOBJECT**) - input
Object handle of the object whose file/directory specification is to be returned.

WinQueryObjectPath Parameter - pszPathname

pszPathname (**PSZ**) - output
Memory allocated by caller in which directory specification is written.

WinQueryObjectPath Parameter - ulSize

ulSize (**ULONG**) - input
Number of bytes pointed to by *pszPathname*.

WinQueryObjectPath Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinQueryObjectPath - Parameters

hObject ([HOBJECT](#)) - input
Object handle of the object whose file/directory specification is to be returned.

pszPathname ([PSZ](#)) - output
Memory allocated by caller in which directory specification is written.

ulSize ([ULONG](#)) - input
Number of bytes pointed to by *pszPathname*.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinQueryObjectPath - Remarks

This function is used to find the file/directory specification of a given object handle.

Using [HOBJECT](#) for .INI files or files in which an application uses a rename/save/delete sequence is not supported.

WinQueryObjectPath - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_PARAMETERS (0x1208)
An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32 768 to +32 767 cannot be converted to a [SHORT](#), and a negative number cannot be converted to a [ULONG](#) or [USHORT](#).

WinQueryObjectPath - Example Code

This example finds the file/directory of the <WP_OS2SYS> object..

```
#define INCL_WINWORKPLACE
#include <os2.h>

HOBJECT hObject;
CHAR      szPath[CCHMAXPATH + 1];
BOOL      fSuccess;

hObject = WinQueryObject("<WP_OS2SYS>");
if (hObject != NULL)
{
    /* WinQueryObject was successful */

    fSuccess = WinQueryObjectPath(hObject, szPath, sizeof(szPath));
    if (fSuccess)
    {
        /* WinQueryObjectPath was successful */
    }
    else
    {
        /* WinQueryObjectPath failed */
    }
}
```

```
else
{
    /* WinQueryObject failed */
}
```

WinQueryObjectPath - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Glossary](#)

WinQueryObjectWindow

WinQueryObjectWindow - Syntax

This function returns the desktop object window handle.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndDesktop; /* Desktop-window handle. */
HWND    hwndObject;  /* Object-window handle. */

hwndObject = WinQueryObjectWindow(hwndDesktop);
```

WinQueryObjectWindow Parameter - hwndDesktop

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP
The desktop-window handle

Other
Specified desktop-window handle.

WinQueryObjectWindow Return Value - hwndObject

hwndObject ([HWND](#)) - returns
Object-window handle.

NULLHANDLE
Error occurred.

WinQueryObjectWindow - Parameters

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP
The desktop-window handle
Other
Specified desktop-window handle.

hwndObject ([HWND](#)) - returns
Object-window handle.

NULLHANDLE
Error occurred.

WinQueryObjectWindow - Remarks

Any window created as a descendant of *hwndObject* is an object window.

WinQueryObjectWindow - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinQueryObjectWindow - Related Functions

Related Functions

- [WinEnableWindow](#)
- [WinIsThreadActive](#)
- [WinIsWindow](#)
- [WinIsWindowEnabled](#)
- [WinQueryDesktopWindow](#)
- [WinQueryObjectWindow](#)
- [WinQueryWindowDC](#)
- [WinQueryWindowProcess](#)

- [WinQueryWindowRect](#)
- [WinWindowFromDC](#)
- [WinWindowFromID](#)
- [WinWindowFromPoint](#)

WinQueryObjectWindow - Example Code

This example calls WinQueryObjectWindow to return the desktop object window handle. All windows created as descendants of this object window-as in the example-will be object windows.

```
#define INCL_WINWINDOWMGR      /* Window Manager Functions */
#include <os2.h>

HWND    hwndObject;           /* desktop object window */
HWND    hwndObject1;          /* descendant object window */
USHORT   WindowId;
hwndObject = WinQueryObjectWindow(HWND_DESKTOP);

/* create object window */
hwndObject1 = WinCreateWindow(hwndObject, /* parent window */
                              "NewClass", /* class name */
                              "new button", /* window text */
                              WS_VISIBLE, /* window style */
                              0, 0, /* position (x,y) */
                              200, 100, /* size (width,height) */
                              0L, /* owner window */
                              HWND_TOP, /* sibling window */
                              WindowId, /* window id */
                              NULL, /* control data */
                              NULL); /* presentation parms */
```

WinQueryObjectWindow - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Errors](#)
- [Remarks](#)
- [Example Code](#)
- [Related Functions](#)
- [Glossary](#)

WinQueryPointer

WinQueryPointer - Syntax

This function returns the pointer handle for *hwndDeskTop*.

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndDesktop; /* Desktop-window handle. */
HPOINTER  hptrPointer; /* Pointer handle. */

hptrPointer = WinQueryPointer(hwndDesktop);
```

WinQueryPointer Parameter - hwndDesktop

hwndDesktop (**HWND**) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

WinQueryPointer Return Value - hptrPointer

hptrPointer (**HPOINTER**) - returns
Pointer handle.

NULLHANDLE	Error occurred.
------------	-----------------

WinQueryPointer - Parameters

hwndDesktop (**HWND**) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

hptrPointer (**HPOINTER**) - returns
Pointer handle.

NULLHANDLE	Error occurred.
------------	-----------------

WinQueryPointer - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinQueryPointer - Related Functions

Related Functions

- [WinCreatePointer](#)
 - [WinCreatePointerIndirect](#)
 - [WinDestroyPointer](#)
 - [WinDrawPointer](#)
 - [WinLoadPointer](#)
 - [WinQueryPointer](#)
 - [WinQueryPointerInfo](#)
 - [WinQueryPointerPos](#)
 - [WinQuerySysPointer](#)
 - [WinQuerySysPointerData](#)
 - [WinSetPointer](#)
 - [WinSetPointerPos](#)
 - [WinSetSysPointerData](#)
 - [WinShowPointer](#)
-

WinQueryPointer - Example Code

This example obtains the pointer handle from the desktop window handle.

```
#define INCL_WINPOINTERS
#define INCL_WINDESKTOP
#include <OS2.H>

HAB      hab;
HPOINTER hpointer;

hpointer = WinQueryPointer(HWND_DESKTOP);
```

WinQueryPointer - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryPointerInfo

WinQueryPointerInfo - Syntax

This function returns pointer information.

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HPOINTER      hptr;          /* Pointer handle. */
PPOINTERINFO  pptriPointerInfo; /* Pointer-information structure. */
BOOL          rc;            /* Success indicator. */

rc = WinQueryPointerInfo(hptr, pptriPointerInfo);
```

WinQueryPointerInfo Parameter - hptr

hptr (**HPOINTER**) - input
Pointer handle.

WinQueryPointerInfo Parameter - pptriPointerInfo

pptriPointerInfo (**PPOINTERINFO**) - output
Pointer-information structure.

WinQueryPointerInfo Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinQueryPointerInfo - Parameters

hptr ([HPOINTER](#)) - input
Pointer handle.

pptriPointerInfo ([PPOINTERINFO](#)) - output
Pointer-information structure.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinQueryPointerInfo - Remarks

The pointer information structure contains information such as the bit-map handle of the pointer and action point coordinates. The values returned for the parameters are in units relative to the size of the system icon or system pointer.

For example, if the application creates a pointer out of a bit map xWide units wide and positions the x-coordinate of the pointer's action point at xHot, then this function will return the value of the as:

```
xHotspot = (xHot * SystemPointerWidth) / xWide
```

where SystemPointerWidth can be obtained by using the [WinQuerySysValue](#) function.

WinQueryPointerInfo - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HPTR (0x101B)
An invalid pointer handle was specified.

WinQueryPointerInfo - Related Functions

Related Functions

- [WinCreatePointer](#)
- [WinCreatePointerIndirect](#)
- [WinDestroyPointer](#)
- [WinDrawPointer](#)
- [WinLoadPointer](#)
- [WinQueryPointer](#)
- [WinQueryPointerInfo](#)
- [WinQueryPointerPos](#)
- [WinQuerySysPointer](#)
- [WinQuerySysPointerData](#)
- [WinSetPointer](#)
- [WinSetPointerPos](#)
- [WinSetSysPointerData](#)
- [WinShowPointer](#)

WinQueryPointerInfo - Example Code

This example uses the WinQueryPointerInfo call to obtain the bit-map handle of the color bit map.

```
#define INCL_WINPOINTERS
#define INCL_WINDESKTOP
#include <OS2.H>

HAB      hab;
HPOINTER hpointer;
POINTERINFO pointerinfo;
HBITMAP  hbm;          /* Bit-map handle of color bit map */

hpointer = WinQueryPointer(HWND_DESKTOP);

WinQueryPointerInfo(hpointer,
                    &pointerinfo);

hbm = pointerinfo.hbmColor;
```

WinQueryPointerInfo - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryPointerPos

WinQueryPointerPos - Syntax

This function returns the pointer position.

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndDeskTop; /* Desktop-window handle. */
PPOINTL   pptlPoint;   /* Pointer position in screen coordinates. */
BOOL      rc;          /* Pointer position returned indicator. */
```

```
rc = WinQueryPointerPos(hwndDeskTop, pptlPoint);
```

WinQueryPointerPos Parameter - hwndDeskTop

hwndDeskTop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

WinQueryPointerPos Parameter - pptlPoint

pptlPoint ([PPOINTL](#)) - output
Pointer position in screen coordinates.

WinQueryPointerPos Return Value - rc

rc ([BOOL](#)) - returns
Pointer position returned indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinQueryPointerPos - Parameters

hwndDeskTop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

pptlPoint ([PPOINTL](#)) - output
Pointer position in screen coordinates.

rc ([BOOL](#)) - returns
Pointer position returned indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinQueryPointerPos - Remarks

The [WinQueryMsgPos](#) is used to get the pointer position of the last message obtained by means of the [WinGetMsg](#) or [WinPeekMsg](#) functions.

WinQueryPointerPos - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinQueryPointerPos - Related Functions

Related Functions

- [WinCreatePointer](#)
- [WinCreatePointerIndirect](#)
- [WinDestroyPointer](#)
- [WinDrawPointer](#)
- [WinLoadPointer](#)
- [WinQueryPointer](#)
- [WinQueryPointerInfo](#)
- [WinQueryPointerPos](#)
- [WinQuerySysPointer](#)
- [WinQuerySysPointerData](#)
- [WinSetPointer](#)
- [WinSetPointerPos](#)
- [WinSetSysPointerData](#)
- [WinShowPointer](#)

WinQueryPointerPos - Example Code

This example displays the pointer position.

```
#define INCL_WINWINDOWMGR
#define INCL_WINPOINTERS
#include <OS2.H>

HWND    hwndClient;
CHAR    szMsg[100];
POINTL  ptl;

WinQueryPointerPos(HWND_DESKTOP, &ptl);
```

```

sprintf(szMsg, "x = %ld y = %ld", ptl.x, ptl.y);
WinMessageBox(HWND_DESKTOP,
              hwndClient,          /* client-window handle */
              szMsg,               /* body of the message */
              "Debugging information", /* title of the message */
              0,                  /* message box id */
              MB_NOICON | MB_OK); /* icon and button flags */

```

WinQueryPointerPos - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryPresParam

WinQueryPresParam - Syntax

This function queries the values of presentation parameters for a window.

```

#define INCL_WINSYS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwnd;          /* Window handle whose presentation is to be queried. */
ULONG     idAttrType1;   /* First presentation parameter to retrieve. */
ULONG     idAttrType2;   /* Second presentation parameter to retrieve. */
PULONG    pidAttrTypeFound; /* Presentation parameter attribute found. */
ULONG     cbAttrValueLen; /* Size of the buffer pointed to by the pAttrValue parameter. */
PPVOID    pAttrValue;    /* Attribute value. */
ULONG     flOptions;      /* Options controlling the query. */
ULONG     cbRetLen;       /* Length of presentation parameter value. passed back. */

cbRetLen = WinQueryPresParam(hwnd, idAttrType1,
                             idAttrType2, pidAttrTypeFound,
                             cbAttrValueLen, pAttrValue, flOptions);

```

WinQueryPresParam Parameter - hwnd

hwnd (HWND) - input
Window handle whose presentation is to be queried.

WinQueryPresParam Parameter - idAttrType1

idAttrType1 (ULONG) - input
First presentation parameter to retrieve.

This identifies the first presentation parameter attribute to be queried. It can be zero to reference no presentation parameter attribute.

WinQueryPresParam Parameter - idAttrType2

idAttrType2 (ULONG) - input
Second presentation parameter to retrieve.

This identifies the second presentation parameter attribute to be queried. It can be zero to reference no presentation parameter attribute.

WinQueryPresParam Parameter - pidAttrTypeFound

pidAttrTypeFound (PULONG) - in/out
Presentation parameter attribute found.

This identifies which of the presentation parameter attributes *idAttrType1* and *idAttrType2* has been found. This parameter can be passed as NULL (if, for example, only one attribute is being queried).

WinQueryPresParam Parameter - cbAttrValueLen

cbAttrValueLen (ULONG) - input
Size of the buffer pointed to by the *pAttrValue* parameter.

WinQueryPresParam Parameter - pAttrValue

pAttrValue (PPVOID) - output
Attribute value.

The value of the presentation parameter attribute found.

WinQueryPresParam Parameter - flOptions

flOptions (ULONG) - input

Options controlling the query.

Any of the following values can be ORed together:

QPF_NOINHERIT

For the purposes of this query, presentation parameters are not inherited from the owners of the window specified by *hwnd*. If not specified (default), presentation parameters are inherited.

QPF_ID1COLORINDEX

idAttrType1 refers to a color index presentation parameter attribute, the value of which, if found, is to be converted to RGB before being passed back in *pAttrValue*.

QPF_ID2COLORINDEX

idAttrType2 refers to a color index presentation parameter attribute, the value of which, if found, is to be converted to RGB before being passed back in *pAttrValue*.

QPF_PURERGBCOLOR

Specifies that either or both of *idAttrType1* and *idAttrType2* reference RGB color, and that these colors must be pure. This is necessary when specifying text foreground and background colors. This is applied after QPF_ID1COLORINDEX and QPF_ID2COLORINDEX.

WinQueryPresParam Return Value - cbRetLen

cbRetLen (ULONG) - returns

Length of presentation parameter value. passed back.

Zero

Presentation parameter not found or error occurred

Other

Length of presentation parameter value passed back in *pAttrValue*.

WinQueryPresParam - Parameters

hwnd (HWND) - input

Window handle whose presentation is to be queried.

idAttrType1 (ULONG) - input

First presentation parameter to retrieve.

This identifies the first presentation parameter attribute to be queried. It can be zero to reference no presentation parameter attribute.

idAttrType2 (ULONG) - input

Second presentation parameter to retrieve.

This identifies the second presentation parameter attribute to be queried. It can be zero to reference no presentation parameter attribute.

pidAttrTypeFound ([PULONG](#)) - in/out

Presentation parameter attribute found.

This identifies which of the presentation parameter attributes *idAttrType1* and *idAttrType2* has been found. This parameter can be passed as NULL (if, for example, only one attribute is being queried).

cbAttrValueLen ([ULONG](#)) - input

Size of the buffer pointed to by the *pAttrValue* parameter.

pAttrValue ([PPVOID](#)) - output

Attribute value.

The value of the presentation parameter attribute found.

flOptions ([ULONG](#)) - input

Options controlling the query.

Any of the following values can be ORed together:

QPF_NOINHERIT

For the purposes of this query, presentation parameters are not inherited from the owners of the window specified by *hwnd*. If not specified (default), presentation parameters are inherited.

QPF_ID1COLORINDEX

idAttrType1 refers to a color index presentation parameter attribute, the value of which, if found, is to be converted to RGB before being passed back in *pAttrValue*.

QPF_ID2COLORINDEX

idAttrType2 refers to a color index presentation parameter attribute, the value of which, if found, is to be converted to RGB before being passed back in *pAttrValue*.

QPF_PURERGBCOLOR

Specifies that either or both of *idAttrType1* and *idAttrType2* reference RGB color, and that these colors must be pure. This is necessary when specifying text foreground and background colors. This is applied after QPF_ID1COLORINDEX and QPF_ID2COLORINDEX.

cbRetLen ([ULONG](#)) - returns

Length of presentation parameter value. passed back.

Zero

Presentation parameter not found or error occurred

Other

Length of presentation parameter value passed back in *pAttrValue*.

WinQueryPresParam - Remarks

Two presentation parameter attribute identities can be passed, and both will be searched for along the chain of owners of the window *hwnd* (subject to QPF_NOINHERIT). The first one found satisfies the query. If both *idAttrType1* and *idAttrType2* are present for the same window, *idAttrType1* takes precedence.

If the presentation parameter attribute value is too long to fit in the *pAttrValue* buffer provided, it is truncated, and the number of bytes copied is returned in *cbRetLen*. (See also [WinSetPresParam](#) and [WinRemovePresParam](#)).

WinQueryPresParam - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)

An invalid window handle was specified.

WinQueryPresParam - Related Functions

Related Functions

- [WinDrawBitmap](#)
 - [WinDrawBorder](#)
 - [WinDrawPointer](#)
 - [WinDrawText](#)
 - [WinFillRect](#)
 - [WinGetSysBitmap](#)
 - [WinInvertRect](#)
 - [WinQueryPresParam](#)
 - [WinRemovePresParam](#)
 - [WinScrollWindow](#)
 - [WinSetPresParam](#)
-

WinQueryPresParam - Example Code

This example queries the disable-foreground attribute; if it is a valid attribute of the window, it is removed via WinRemovePresParam.

```
#define INCL_WINSYS
#include <OS2.H>

HWND  hwnd;
ULONG AttrFound;
ULONG AttrValue[32];
ULONG cbRetLen;

cbRetLen = WinQueryPresParam(hwnd,
                             PP_DISABLEDFOREGROUNDCOLORINDEX,
                             0,
                             &AttrFound,
                             sizeof(AttrValue),
                             &AttrValue,
                             QPF_ID1COLORINDEX | QPF_NOINHERIT);

if(PP_DISABLEDFOREGROUNDCOLORINDEX == AttrFound)

WinRemovePresParam(hwnd,
                    PP_DISABLEDFOREGROUNDCOLORINDEX);
```

WinQueryPresParam - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryQueueInfo

WinQueryQueueInfo - Syntax

This function returns the information for the specified queue.

```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HMQ      hmq;          /* Queue handle. */
PMQINFO  pmqiMqinfo;   /* Message queue information structure to contain the queue information. */
ULONG    cbCopied;     /* Size of message queue information structure that is provided (in bytes). */
BOOL     rc;           /* Success indicator. */

rc = WinQueryQueueInfo(hmq, pmqiMqinfo, cbCopied);
```

WinQueryQueueInfo Parameter - hmq

hmq (**HMQ**) - input
Queue handle.

It must be created by a previous call to [WinCreateMsgQueue](#) or HMQ_CURRENT.

WinQueryQueueInfo Parameter - pmqiMqinfo

pmqiMqinfo (**PMQINFO**) - output
Message queue information structure to contain the queue information.

WinQueryQueueInfo Parameter - cbCopied

cbCopied (**ULONG**) - input
Size of message queue information structure that is provided (in bytes).

Specifies the maximum number of bytes to be copied into the *pmqiMqinfo* parameter. This should be the size of an **MQINFO** structure.

WinQueryQueueInfo Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinQueryQueueInfo - Parameters

hmq ([HMQ](#)) - input
Queue handle.

It must be created by a previous call to [WinCreateMsgQueue](#) or HMQ_CURRENT.

pmqiMqinfo ([PMQINFO](#)) - output
Message queue information structure to contain the queue information.

cbCopied ([ULONG](#)) - input
Size of message queue information structure that is provided (in bytes).

Specifies the maximum number of bytes to be copied into the *pmqiMqinfo* parameter. This should be the size of an [MQINFO](#) structure.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinQueryQueueInfo - Related Functions

Related Functions

- [WinBroadcastMsg](#)
- [WinCreateMsgQueue](#)
- [WinDestroyMsgQueue](#)
- [WinDispatchMsg](#)
- [WinGetDlgMsg](#)
- [WinGetMsg](#)
- [WinInSendMessage](#)
- [WinPeekMsg](#)
- [WinPostMsg](#)
- [WinPostQueueMsg](#)
- [WinQueryMsgPos](#)
- [WinQueryMsgTime](#)
- [WinQueryQueueInfo](#)
- [WinQueryQueueStatus](#)
- [WinSendDlgItemMsg](#)
- [WinSendMessage](#)

- [WinSetClassMsgInterest](#)
- [WinSetMsgInterest](#)
- [WinSetMsgMode](#)
- [WinSetSynchroMode](#)
- [WinWaitMsg](#)

WinQueryQueueInfo - Example Code

This example retrieves the process identity from a queue by passing the queue handle to WinQueryQueueInfo.

```
#define INCL_WINMESSAGEMGR
#include <OS2.H>

HMQ      hmq;
MQINFO   mqinfo;
PID       pid;

WinQueryQueueInfo(hmq,
                  &mqinfo,
                  sizeof(MQINFO));

pid = mqinfo.pid;
```

WinQueryQueueInfo - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryQueueStatus

WinQueryQueueStatus - Syntax

This function returns a code indicating the status of the message queue associated with the caller.

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndDesktop; /* Desktop-window handle. */
ULONG     flStatus;     /* Status information. */

flStatus = WinQueryQueueStatus(hwndDesktop);
```

WinQueryQueueStatus Parameter - hwndDesktop

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

[HWND_DESKTOP](#)
The desktop-window handle

Other
Desktop-window handle returned by [WinQueryDesktopWindow](#).

WinQueryQueueStatus Return Value - flStatus

flStatus ([ULONG](#)) - returns
Status information.

Summary

- Summary of message types existing on the queue.
- This field contains a combination of the following values:
- QS_KEY**
An input event (keyboard or journaling) has caused a [WM_CHAR](#) message to be placed in the queue.
 - QS_MOUSE**
An input event has caused a [WM_MOUSEMOVE](#), [WM_BUTTON1UP](#), [WM_BUTTON1DOWN](#), [WM_BUTTON1DBLCLK](#), [WM_BUTTON2UP](#), [WM_BUTTON2DOWN](#), [WM_BUTTON2DBLCLK](#), [WM_BUTTON3UP](#), [WM_BUTTON3DOWN](#), or [WM_BUTTON3DBLCLK](#) message to be placed in the queue.
 - QS_MOUSEBUTTON**
An input event has caused a [WM_BUTTON1UP](#), [WM_BUTTON1DOWN](#), [WM_BUTTON1DBLCLK](#), [WM_BUTTON2UP](#), [WM_BUTTON2DOWN](#), [WM_BUTTON2DBLCLK](#), [WM_BUTTON3UP](#), [WM_BUTTON3DOWN](#), or [WM_BUTTON3DBLCLK](#) message to be placed in the queue.
 - QS_MOUSEMOVE**
An input event has caused a [WM_MOUSEMOVE](#) message to be placed in the queue.
 - QS_TIMER**
A timer event has caused a [WM_TIMER](#) message to be placed in the queue.
 - QS_PAINT**
A [WM_PAINT](#) message is available.
 - QS_SEM1**
A [WM_SEM1](#) message is available.
 - QS_SEM2**
A [WM_SEM2](#) message is available.
 - QS_SEM3**
A [WM_SEM3](#) message is available.
 - QS_SEM4**
A [WM_SEM4](#) message is available.

QS_POSTMSG	A message has been posted to the queue. Note that this message is probably not one of the messages listed above, but could be a WM_CHAR , WM_MOUSEMOVE or similar message if an application has posted one of these. In this case, the corresponding input status flag (QS_KEY, QS_MOUSE, and so on) is not set.
QS_SENDMSG	A message has been sent by another application to a window associated with the current queue.

Added

Message type additions.

Message types added to the queue since the last use of this function. The value of this field is a subset of the *Summary* field.

WinQueryQueueStatus - Parameters

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP

The desktop-window handle

Other

Desktop-window handle returned by [WinQueryDesktopWindow](#).

flStatus ([ULONG](#)) - returns
Status information.

Summary

Summary of message types existing on the queue.

This field contains a combination of the following values:

QS_KEY	An input event (keyboard or journaling) has caused a WM_CHAR message to be placed in the queue.
QS_MOUSE	An input event has caused a WM_MOUSEMOVE , WM_BUTTON1UP , WM_BUTTON1DOWN , WM_BUTTON1DBLCLK , WM_BUTTON2UP , WM_BUTTON2DOWN , WM_BUTTON2DBLCLK , WM_BUTTON3UP , WM_BUTTON3DOWN , or WM_BUTTON3DBLCLK message to be placed in the queue.
QS_MOUSEBUTTON	An input event has caused a WM_BUTTON1UP , WM_BUTTON1DOWN , WM_BUTTON1DBLCLK , WM_BUTTON2UP , WM_BUTTON2DOWN , WM_BUTTON2DBLCLK , WM_BUTTON3UP , WM_BUTTON3DOWN , or WM_BUTTON3DBLCLK message to be placed in the queue.
QS_MOUSEMOVE	An input event has caused a WM_MOUSEMOVE message to be placed in the queue.
QS_TIMER	A timer event has caused a WM_TIMER message to be placed in the queue.
QS_PAINT	A WM_PAINT message is available.
QS_SEM1	A WM_SEM1 message is available.
QS_SEM2	A WM_SEM2 message is available.

QS_SEM3	A WM_SEM3 message is available.
QS_SEM4	A WM_SEM4 message is available.
QS_POSTMSG	A message has been posted to the queue. Note that this message is probably not one of the messages listed above, but could be a WM_CHAR , WM_MOUSEMOVE or similar message if an application has posted one of these. In this case, the corresponding input status flag (QS_KEY, QS_MOUSE, and so on) is not set.
QS_SENDMSG	A message has been sent by another application to a window associated with the current queue.

Added

Message type additions.

Message types added to the queue since the last use of this function. The value of this field is a subset of the *Summary* field.

WinQueryQueueStatus - Remarks

This function is an efficient method for determining whether input is available for processing by the [WinGetMsg](#) or [WinPeekMsg](#) functions.

WinQueryQueueStatus - Related Functions

Related Functions

- [WinBroadcastMsg](#)
- [WinCreateMsgQueue](#)
- [WinDestroyMsgQueue](#)
- [WinDispatchMsg](#)
- [WinGetDlgMsg](#)
- [WinGetMsg](#)
- [WinInSendMessage](#)
- [WinPeekMsg](#)
- [WinPostMsg](#)
- [WinPostQueueMsg](#)
- [WinQueryMsgPos](#)
- [WinQueryMsgTime](#)
- [WinQueryQueueInfo](#)
- [WinQueryQueueStatus](#)
- [WinSendDlgItemMsg](#)
- [WinSendMessage](#)
- [WinSetClassMsgInterest](#)
- [WinSetMsgInterest](#)
- [WinSetMsgMode](#)
- [WinSetSynchroMode](#)
- [WinWaitMsg](#)

WinQueryQueueStatus - Related Messages

Related Messages

- [WM_BUTTON1UP](#)
- [WM_BUTTON1DOWN](#)
- [WM_BUTTON1DBLCLK](#)
- [WM_BUTTON2UP](#)
- [WM_BUTTON2DOWN](#)
- [WM_BUTTON2DBLCLK](#)
- [WM_BUTTON3UP](#)
- [WM_BUTTON3DOWN](#)
- [WM_BUTTON3DBLCLK](#)
- [WM_CHAR](#)
- [WM_MOUSEMOVE](#)
- [WM_PAINT](#)
- [WM_SEM1](#)
- [WM_SEM2](#)
- [WM_SEM3](#)
- [WM_SEM4](#)
- [WM_TIMER](#)

WinQueryQueueStatus - Example Code

This example uses the WinQueryQueueStatus to see if a WM_MOUSEMOVE message has been placed in the queue.

```
#define INCL_WINMESSAGEGR
#include <OS2.H>

if (WinQueryQueueStatus (HWND_DESKTOP) == QS_MOUSEMOVE)
{
    /* . */
    /* . */
}
```

WinQueryQueueStatus - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinQuerySessionTitle

WinQuerySessionTitle - Syntax

This function obtains the title under which a specified application is started, or is added to the Window List.

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
ULONG    ulSession;    /* Session identity of application whose title is requested. */
PSZ      pszTitle;     /* Window List title. */
ULONG    ulTitlelen;   /* Maximum length of data returnable, in bytes. */
ULONG    rc;           /* Return code. */

rc = WinQuerySessionTitle(hab, ulSession,
                          pszTitle, ulTitlelen);
```

WinQuerySessionTitle Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinQuerySessionTitle Parameter - ulSession

ulSession (**ULONG**) - input
Session identity of application whose title is requested.

- | | |
|-------|--|
| 0 | Use the session identity of the caller |
| Other | Use the specified session identity. |

WinQuerySessionTitle Parameter - pszTitle

pszTitle (**PSZ**) - output
Window List title.

This is the title of the application with a process identity, if the application is present in the Window List.

WinQuerySessionTitle Parameter - ulTitlelen

ulTitlelen (**ULONG**) - input
Maximum length of data returnable, in bytes.

If the *pszTitle* parameter is longer than this value, the title is truncated. However, the terminating null character is left at the end of the string. The maximum number of title characters copied is (*ulTitlelen*-1).

WinQuerySessionTitle Return Value - rc

rc ([ULONG](#)) - returns
Return code.

0	Successful completion
Other	Error occurred.

WinQuerySessionTitle - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

ulSession ([ULONG](#)) - input
Session identity of application whose title is requested.

0	Use the session identity of the caller
Other	Use the specified session identity.

pszTitle ([PSZ](#)) - output
Window List title.

This is the title of the application with a process identity, if the application is present in the Window List.

ulTitlelen ([ULONG](#)) - input
Maximum length of data returnable, in bytes.

If the *pszTitle* parameter is longer than this value, the title is truncated. However, the terminating null character is left at the end of the string. The maximum number of title characters copied is (*ulTitlelen*-1).

rc ([ULONG](#)) - returns
Return code.

0	Successful completion
Other	Error occurred.

WinQuerySessionTitle - Remarks

This function is useful when an application uses the same name in its window title (and in its entry in the Window List) as the end user invokes to start the application. This provides a visual link for the end user.

If this function is used after a Window List entry is created for the application, the title in the Window List entry is obtained. (See also [WinQueryTaskTitle](#).)

WinQuerySessionTitle - Related Functions

Related Functions

- [WinAddSwitchEntry](#)
- [WinChangeSwitchEntry](#)
- [WinCreateSwitchEntry](#)
- [WinQuerySessionTitle](#)
- [WinQuerySwitchEntry](#)
- [WinQuerySwitchHandle](#)
- [WinQuerySwitchList](#)
- [WinQueryTaskSizePos](#)
- [WinQueryTaskTitle](#)
- [WinRemoveSwitchEntry](#)
- [WinSwitchToProgram](#)

WinQuerySessionTitle - Example Code

This example calls WinQuerySessionTitle to retrieve the application's title, and then sets the title bar of the frame window to that title.

```
#define INCL_WINMESSAGEGR
#define INCL_WINWINDOWMGR
#include <OS2.H>

HAB hab;
HWND hwndFrame, hwndClient;
CHAR szTitle[MAXNAMEL + 1];

WinQuerySessionTitle(hab,
                    0,
                    szTitle,
                    sizeof(szTitle));

hwndFrame = WinQueryWindow(hwndClient,
                          QW_PARENT); /* get handle of parent, */
                                      /* which is frame window. */
WinSetWindowText(hwndFrame, szTitle);
```

WinQuerySessionTitle - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQuerySwitchEntry

WinQuerySwitchEntry - Syntax

This function obtains a copy of the Window List data for a specific application.

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HSWITCH      hswitchSwitch; /* Handle to the Window List entry. */
PSWCNTRL     pswctlSwitchData; /* Switch control data. */
ULONG        rc; /* Return code. */

rc = WinQuerySwitchEntry(hswitchSwitch, pswctlSwitchData);
```

WinQuerySwitchEntry Parameter - hswitchSwitch

hswitchSwitch ([HSWITCH](#)) - input
Handle to the Window List entry.

This can be obtained using the [WinQuerySwitchHandle](#) function.

WinQuerySwitchEntry Parameter - pswctlSwitchData

pswctlSwitchData ([PSWCNTRL](#)) - output
Switch control data.

Contains information about the specified Window List entry. The field contains the program handle used to start the program.

WinQuerySwitchEntry Return Value - rc

rc ([ULONG](#)) - returns
Return code.

0	Successful completion
Other	Error occurred.

WinQuerySwitchEntry - Parameters

hswitchSwitch ([HSWITCH](#)) - input
Handle to the Window List entry.

This can be obtained using the [WinQuerySwitchHandle](#) function.

pswctlSwitchData ([PSWCNTRL](#)) - output
Switch control data.

Contains information about the specified Window List entry. The field contains the program handle used to start the program.

rc ([ULONG](#)) - returns
Return code.

0	Successful completion
Other	Error occurred.

WinQuerySwitchEntry - Remarks

This function is available to PM and non-PM applications.

WinQuerySwitchEntry - Related Functions

Related Functions

- [WinAddSwitchEntry](#)
- [WinChangeSwitchEntry](#)
- [WinCreateSwitchEntry](#)
- [WinQuerySessionTitle](#)
- [WinQuerySwitchEntry](#)
- [WinQuerySwitchHandle](#)
- [WinQuerySwitchList](#)
- [WinQueryTaskSizePos](#)
- [WinQueryTaskTitle](#)
- [WinRemoveSwitchEntry](#)
- [WinSwitchToProgram](#)

WinQuerySwitchEntry - Example Code

This example calls [WinQuerySwitchHandle](#) to get the Task List handle of a frame window, and then calls [WinQuerySwitchEntry](#) to retrieve information about that application.

```
#define INCL_WINSWITCHLIST
#include <OS2.H>

HAB      hab;
HWND     hwndFrame;
HSWITCH  hswitch;
```

```
SWCCTRL swctl;

hswitch = WinQuerySwitchHandle(hwndFrame, 0);
WinQuerySwitchEntry(hswitch, &swctl);
```

WinQuerySwitchEntry - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQuerySwitchHandle

WinQuerySwitchHandle - Syntax

This function obtains the Window List handle belonging to a window.

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwnd;          /* Window handle of an application. */
PID       idProcess;     /* Process identity of the application. */
HSWITCH   hswitchSwitch; /* Switch list handle for the specified application. */

hswitchSwitch = WinQuerySwitchHandle(hwnd,
                                     idProcess);
```

WinQuerySwitchHandle Parameter - hwnd

hwnd (**HWND**) - input

Window handle of an application.

Window handle of an application running in the OS/2 session for which the Window List handle is required.

NULLHANDLE

Application is not an OS/2 application

Other

Window handle of an application.

WinQuerySwitchHandle Parameter - idProcess

idProcess (**PID**) - input
Process identity of the application.

WinQuerySwitchHandle Return Value - hswitchSwitch

hswitchSwitch (**HSWITCH**) - returns
Switch list handle for the specified application.

NULLHANDLE	Application is not in the switch list, or an error occurred
Other	Switch list handle.

WinQuerySwitchHandle - Parameters

hwnd (**HWND**) - input
Window handle of an application.

Window handle of an application running in the OS/2 session for which the Window List handle is required.

NULLHANDLE	Application is not an OS/2 application
Other	Window handle of an application.

idProcess (**PID**) - input
Process identity of the application.

hswitchSwitch (**HSWITCH**) - returns
Switch list handle for the specified application.

NULLHANDLE	Application is not in the switch list, or an error occurred
Other	Switch list handle.

WinQuerySwitchHandle - Remarks

If both a window handle and a process identity are supplied, they must be consistent.

If the window handle is NULL and the process identity supplied cannot be found in the switch list then the switch handle returned is the handle for the most proximal ancestor process. Once the switch list handle is obtained, it may be used in various other calls to manipulate the switch list entry or the program which it references.

WinQuerySwitchHandle - Related Functions

Related Functions

- [WinAddSwitchEntry](#)
 - [WinChangeSwitchEntry](#)
 - [WinCreateSwitchEntry](#)
 - [WinQuerySessionTitle](#)
 - [WinQuerySwitchEntry](#)
 - [WinQuerySwitchHandle](#)
 - [WinQuerySwitchList](#)
 - [WinQueryTaskSizePos](#)
 - [WinQueryTaskTitle](#)
 - [WinRemoveSwitchEntry](#)
 - [WinSwitchToProgram](#)
-

WinQuerySwitchHandle - Example Code

This example calls WinQuerySwitchHandle to get the Task List handle of a frame window, and then calls WinQuerySwitchEntry to retrieve information about that application.

```
#define INCL_WINSWITCHLIST
#include <OS2.H>

HAB      hab;
HWND     hwndFrame;
HSWITCH  hswitch;
SWCNTRL  swctl;

hswitch = WinQuerySwitchHandle(hwndFrame, 0);
WinQuerySwitchEntry(hswitch, &swctl);
```

WinQuerySwitchHandle - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQuerySwitchList

WinQuerySwitchList - Syntax

This function obtains information about the entries in the Window List.

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
PSWBLOCK pswblkBlock;  /* Switch entries block. */
ULONG    ulLength;     /* Maximum length of data returnable in bytes. */
ULONG    ulCount;      /* Total number of switch list entries present in the system. */

ulCount = WinQuerySwitchList(hab, pswblkBlock,
                             ulLength);
```

WinQuerySwitchList Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinQuerySwitchList Parameter - pswblkBlock

pswblkBlock (**PSWBLOCK**) - in/out
Switch entries block.

Contains a description of all the entries in the current switch list. This is held in a **SWBLOCK** structure, which has a count of the number of switch list entries, plus a record for each entry containing data such as the process and session identities, the icon handle, and the window handle for the running program.

NULL
No information returned; the return parameter however contains the total number of switch list entries.

Other
Switch entries block.

WinQuerySwitchList Parameter - ulLength

ulLength (**ULONG**) - input
Maximum length of data returnable in bytes.

This is the maximum length in bytes of the data that can be returned in the *pswblkBlock* parameter.

0
No information returned, however the return parameter contains the total number of switch list entries.

Other

Maximum length of data returnable.

WinQuerySwitchList Return Value - ulCount

ulCount ([ULONG](#)) - returns

Total number of switch list entries present in the system.

0

Error occurred

Other

Total number of switch list entries present in the system.

WinQuerySwitchList - Parameters

hab ([HAB](#)) - input

Anchor-block handle.

pswblkBlock ([PSWBLOCK](#)) - in/out

Switch entries block.

Contains a description of all the entries in the current switch list. This is held in a [SWBLOCK](#) structure, which has a count of the number of switch list entries, plus a record for each entry containing data such as the process and session identities, the icon handle, and the window handle for the running program.

NULL

No information returned; the return parameter however contains the total number of switch list entries.

Other

Switch entries block.

ulLength ([ULONG](#)) - input

Maximum length of data returnable in bytes.

This is the maximum length in bytes of the data that can be returned in the *pswblkBlock* parameter.

0

No information returned, however the return parameter contains the total number of switch list entries.

Other

Maximum length of data returnable.

ulCount ([ULONG](#)) - returns

Total number of switch list entries present in the system.

0

Error occurred

Other

Total number of switch list entries present in the system.

WinQuerySwitchList - Remarks

It is possible to obtain information about all the programs currently executing in a single operation, with one array entry for each program.

WinQuerySwitchList - Related Functions

Related Functions

- [WinAddSwitchEntry](#)
 - [WinChangeSwitchEntry](#)
 - [WinCreateSwitchEntry](#)
 - [WinQuerySwitchEntry](#)
 - [WinQuerySwitchHandle](#)
 - [WinRemoveSwitchEntry](#)
 - [WinSwitchToProgram](#)
-

WinQuerySwitchList - Example Code

This example determines the number of items in the Task List, allocates memory for the required buffer, and then calls WinQuerySwitchList again to fill the buffer with the information about each item in the Task List.

```
#define INCL_DOSMEMMGR
#define INCL_DOSERRORS
#define INCL_WINSWITCHLIST /* Or INCL_WIN, INCL_PM */
#include <OS2.H>

#define MEMPOOL 40000 /* Size of local memory pool area */

HAB      hab      = NULLHANDLE;
HWND     hwndFrame = NULLHANDLE;
ULONG    cbItems   = 0,          /* Number of items in list */
          ulBufSize = 0;          /* Size of buffer for information */
PVOID    pvBase    = NULL;       /* Pointer to local memory pool */
PSWBLOCK pswblk    = NULL;       /* Pointer to information returned */
APIRET   rc        = NO_ERROR;   /* Return code from Dos APIs */

/* Allocate a large block of memory (uncommitted) and make
   it available for suballocation. This allows the system
   to commit memory only when it is actually needed. */

rc = DosAllocMem( &pvBase, MEMPOOL, PAG_READ | PAG_WRITE );
rc = DosSubSetMem( pvBase, DOSSUB_INIT | DOSSUB_SPARSE_OBJ, MEMPOOL );

/* Determine the number of items in the list and calculate
   the size of the buffer needed. */

cbItems = WinQuerySwitchList(hab, NULL, 0);
ulBufSize = (cbItems * sizeof(SWENTRY)) + sizeof(HSWITCH);

/* Allocate the buffer from our memory pool */

rc = DosSubAllocMem( pvBase, (PVOID) &pswblk, ulBufSize);

/* Call WinQuerySwitchList again to fill our buffer with information */

cbItems = WinQuerySwitchList(hab, pswblk, ulBufSize);
```

WinQuerySwitchList - Topics

Select an item:

[Syntax](#)

- Parameters
- Returns
- Remarks
- Example Code
- Related Functions
- Glossary

WinQuerySysColor

WinQuerySysColor - Syntax

This function returns the system color.

```
#define INCL_WINSYS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndDesktop; /* Desktop-window handle. */
LONG    clr;          /* System color-index value. */
LONG    lReserved;    /* Reserved value, must be 0. */
LONG    lRgbColor;    /* RGB value. */

lRgbColor = WinQuerySysColor(hwndDesktop,
                             clr, lReserved);
```

WinQuerySysColor Parameter - hwndDesktop

hwndDesktop (**HWND**) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

WinQuerySysColor Parameter - clr

clr (**LONG**) - input
System color-index value.

Must be one of the SYSCLR_* index values.

WinQuerySysColor Parameter - IReserved

IReserved ([LONG](#)) - input
Reserved value, must be 0.

WinQuerySysColor Return Value - IRgbColor

IRgbColor ([LONG](#)) - returns
RGB value.

i1.RGB (red-green-blue) RGB value corresponding to the *clr* parameter.

WinQuerySysColor - Parameters

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

 HWND_DESKTOP The desktop-window handle
 Other Specified desktop-window handle.

clr ([LONG](#)) - input
System color-index value.

 Must be one of the SYSCLR_* index values.

IReserved ([LONG](#)) - input
Reserved value, must be 0.

IRgbColor ([LONG](#)) - returns
RGB value.

i1.RGB (red-green-blue) RGB value corresponding to the *clr* parameter.

WinQuerySysColor - Remarks

This function returns the color value that corresponds to the specified color index of the specified color palette.

The following are the available SYSCLR_* values:

System Color Index	Default Color	Default RGB Values
SYSCLR_ACTIVEBORDER	Pale yellow	255 255 128

SYSCLR_ACTIVETITLE	Teal	64	128	128
SYSCLR_ACTIVETITLETEXT	White	255	255	255
SYSCLR_ACTIVETITLETEXTBGND	Teal	64	128	128
SYSCLR_APPWORKSPACE	Off-white	255	255	224
SYSCLR_BACKGROUND	Light gray	204	204	204
SYSCLR_BUTTONDARK	Dark gray	128	128	128
SYSCLR_BUTTONDEFAULT	Black	0	0	0
SYSCLR_BUTTONLIGHT	White	255	255	255
SYSCLR_BUTTONMIDDLE	Light gray	204	204	204
SYSCLR_DIALOGBACKGROUND	Light gray	204	204	204
SYSCLR_ENTRYFIELD	Pale yellow	255	255	204
SYSCLR_FIELDBACKGROUND	Light gray	204	204	204
SYSCLR_HELPBACKGROUND	White	255	255	255
SYSCLR_HELPHILITE	Blue green	0	128	128
SYSCLR_HELPTEXT	Dark blue	0	0	128
SYSCLR_HILITEBACKGROUND	Dark gray	128	128	128
SYSCLR_HILITEFOREGROUND	White	255	255	255
SYSCLR_ICONTEXT	Black	0	0	0
SYSCLR_INACTIVEBORDER	Light gray	204	204	204
SYSCLR_INACTIVETITLE	Light gray	204	204	204
SYSCLR_INACTIVETITLETEXT	Dark gray	128	128	128
SYSCLR_INACTIVETITLETEXTBGND	Light gray	204	204	204
SYSCLR_MENU	Light gray	204	204	204
SYSCLR_MENUDISABLEDTEXT	Dark gray	128	128	128
SYSCLR_MENUHILITE	Black	0	0	0
SYSCLR_MENUHILITEBGND	Light grey	204	204	204
SYSCLR_MENUTEXT	Black	0	0	0
SYSCLR_OUTPUTTEXT	Black	0	0	0
SYSCLR_PAGEBACKGROUND	White	255	255	255
SYSCLR_SCROLLBAR	Pale gray	192	192	192
SYSCLR_SHADOW	Dark gray	128	128	128
SYSCLR_SHADOWHILITEBGND	Dark gray	128	128	128
SYSCLR_SHADOWHILITEFGND	White	255	255	255
SYSCLR_SHADOWTEXT	Dark gray	128	128	128
SYSCLR_TITLEBOTTOM	Dark gray	128	128	128
SYSCLR_TITLETEXT	White	255	255	255
SYSCLR_WINDOW	White	255	255	255
SYSCLR_WINDOWFRAME	Dark gray	128	128	128
SYSCLR_WINDOWSTATICTEXT	Blue	0	0	128
SYSCLR_WINDOWTEXT	Black	0	0	0

WinQuerySysColor - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_PARAMETER_OUT_OF_RANGE (0x1003)
The value of a parameter was not within the defined valid range for that parameter.

WinQuerySysColor - Related Functions

Related Functions

- [WinQuerySysColor](#)
 - [WinSetSysColors](#)
-

WinQuerySysColor - Example Code

This example uses the WinQuerySysColor to find the RGB index of the system push button, SYSCLR_BUTTONDEFAULT.

```
#define INCL_WINSYS
#define INCL_WINDESKTOP
#include <OS2.H>

HAB hab;
LONG lRgbColor;

lRgbColor = WinQuerySysColor(HWND_DESKTOP,
                             SYSCLR_BUTTONDEFAULT,
                             0L);
```

WinQuerySysColor - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQuerySysModalWindow

WinQuerySysModalWindow - Syntax

This function returns the current system modal window.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndDesktop; /* Desktop-window handle. */
HWND    hwndSysModal; /* Handle of system modal window. */

hwndSysModal = WinQuerySysModalWindow(hwndDesktop);
```

WinQuerySysModalWindow Parameter - hwndDesktop

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

WinQuerySysModalWindow Return Value - hwndSysModal

hwndSysModal ([HWND](#)) - returns
Handle of system modal window.

NULLHANDLE	No system modal window
Other	Handle of system modal window.

WinQuerySysModalWindow - Parameters

hwndDesktop ([HWND](#)) - input

Desktop-window handle.

HWND_DESKTOP

The desktop-window handle

Other

Specified desktop-window handle.

hwndSysModal ([HWND](#)) - returns

Handle of system modal window.

NULLHANDLE

No system modal window

Other

Handle of system modal window.

WinQuerySysModalWindow - Remarks

For a full description of the operation of the system modal window, see the [WinSetSysModalWindow](#) function.

WinQuerySysModalWindow - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)

An invalid window handle was specified.

WinQuerySysModalWindow - Related Functions

Related Functions

- [WinQuerySysModalWindow](#)
- [WinSetSysModalWindow](#)

WinQuerySysModalWindow - Example Code

This example uses the WinQuerySysModalWindow to find the handle of the system modal window.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HAB hab;
HWND hwndDeskTop, hwndSysModal;
LONG lRgbColor;

/* Input processing can enter a "system modal" state. In */
/* this state, all pointing device and keyboard input */
/* is directed to a special window, known as the */
/* system-modal window. Typically, this will be a dialog */
/* window requiring input. */
```

```
hwndSysModal = WinQuerySysModalWindow(hwndDesktop);
```

WinQuerySysModalWindow - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQuerySysPointer

WinQuerySysPointer - Syntax

This function returns the system-pointer handle.

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */  
#include <os2.h>  
  
HWND      hwndDesktop; /* Desktop-window handle. */  
LONG      lIdentifier; /* System-pointer identifier. */  
BOOL      fCopy;       /* Copy indicator. */  
HPOINTER  hptrPointer; /* Pointer handle. */  
  
hptrPointer = WinQuerySysPointer(hwndDesktop,  
                                lIdentifier, fCopy);
```

WinQuerySysPointer Parameter - hwndDesktop

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

WinQuerySysPointer Parameter - lIdentifier

Identifier (LONG) - input
System-pointer identifier.

SPTR_ARROW	Arrow pointer
SPTR_TEXT	Text I-beam pointer
SPTR_WAIT	Hourglass pointer
SPTR_SIZE	Size pointer
SPTR_MOVE	Move pointer
SPTR_SIZENWSE	Downward-sloping, double-headed arrow pointer
SPTR_SIZENESW	Upward-sloping, double-headed arrow pointer
SPTR_SIZewe	Horizontal, double-headed arrow pointer
SPTR_SIZENS	Vertical, double-headed arrow pointer
SPTR_APPICON	Standard application icon pointer
SPTR_ICONINFORMATION	Information icon pointer
SPTR_ICONQUESTION	Question mark icon pointer
SPTR_ICONERROR	Exclamation mark icon pointer
SPTR_ICONWARNING	Warning icon pointer
SPTR_ILLEGAL	Illegal operation icon pointer
SPTR_FILE	Single file icon pointer
SPTR_MULTIFILE	Multiple files icon pointer
SPTR_FOLDER	Folder icon pointer
SPTR_PROGRAM	Application program icon pointer

WinQuerySysPointer Parameter - fCopy

fCopy (BOOL) - input
Copy indicator.

TRUE	Create a copy of the default system pointer and return its handle. Specify this value if the system pointer is to be modified. The application should destroy the copy of the pointer created. This can be done by using the WinDestroyPointer function.
FALSE	Return the handle of the current system pointer.

WinQuerySysPointer Return Value - hptrPointer

hpPtrPointer (**HPOINTER**) - returns
Pointer handle.

WinQuerySysPointer - Parameters

hwndDeskTop (**HWND**) - input
Desktop-window handle.

lIdentifier (**LONG**) - input
System-pointer identifier.

- SPTR_ARROW
Arrow pointer
- SPTR_TEXT
Text I-beam pointer
- SPTR_WAIT
Hourglass pointer
- SPTR_SIZE
Size pointer
- SPTR_MOVE
Move pointer
- SPTR_SIZENWSE
Downward-sloping, double-headed arrow pointer
- SPTR_SIZENESW
Upward-sloping, double-headed arrow pointer
- SPTR_SIZewe
Horizontal, double-headed arrow pointer
- SPTR_SIZENS
Vertical, double-headed arrow pointer
- SPTR_APPICON
Standard application icon pointer
- SPTR_ICONINFORMATION
Information icon pointer
- SPTR_ICONQUESTION
Question mark icon pointer
- SPTR_ICONERROR
Exclamation mark icon pointer
- SPTR_ICONWARNING
Warning icon pointer
- SPTR_ILLEGAL
Illegal operation icon pointer
- SPTR_FILE
Single file icon pointer
- SPTR_MULTIFILE
Multiple files icon pointer
- SPTR_FOLDER
Folder icon pointer
- SPTR_PROGRAM
Application program icon pointer

fCopy (**BOOL**) - input
Copy indicator.

- TRUE
Create a copy of the default system pointer and return its handle. Specify this value if the system pointer is to be modified. The application should destroy the copy of the pointer created. This can be done by using the WinDestroyPointer function.
- FALSE
Return the handle of the current system pointer.

hpPtrPointer (**HPOINTER**) - returns
Pointer handle.

WinQuerySysPointer - Remarks

Take care when using the pointer bit-map handles returned by the [WinQueryPointerInfo](#) function in the [POINTERINFO](#) structure. If the handle is a system-pointer handle, or is returned by the [WinQueryPointerInfo](#) function, it is possible that another application is also accessing the bit-map handle. If this is so, selecting the bit map into a presentation space may fail. Only the active thread may use the bit-map handle returned by either the [WinQuerySysPointer](#) function, when *fCopy* is FALSE, or by the [WinQueryPointerInfo](#) function.

Note: This rule is not enforced by the system; therefore, ensure that the program handles selection failures correctly.

WinQuerySysPointer - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_PARAMETER_OUT_OF_RANGE (0x1003)
The value of a parameter was not within the defined valid range for that parameter.

WinQuerySysPointer - Related Functions

Related Functions

- [WinCreatePointer](#)
 - [WinCreatePointerIndirect](#)
 - [WinDestroyPointer](#)
 - [WinDrawPointer](#)
 - [WinLoadPointer](#)
 - [WinQueryPointer](#)
 - [WinQueryPointerInfo](#)
 - [WinQueryPointerPos](#)
 - [WinQuerySysPointer](#)
 - [WinQuerySysPointerData](#)
 - [WinSetPointer](#)
 - [WinSetPointerPos](#)
 - [WinSetSysPointerData](#)
 - [WinShowPointer](#)
-

WinQuerySysPointer - Example Code

This example calls [WinQuerySysPointer](#) to get a handle to the system pointer, and then loads an application-defined pointer. After it has finished using the application-defined pointer, it restores the system pointer.

```
#define INCL_WINPOINTERS
#include <OS2.H>
#define IDP_CROSSHAIR 900

HWND hptrDefault, hptrCrossHair;
```

```

/* get the system pointer */
hptrDefault = WinQuerySysPointer(HWND_DESKTOP, SPTR_ARROW, FALSE);

/* load an application-defined pointer */
hptrCrossHair = WinLoadPointer(HWND_DESKTOP, (ULONG)0, IDP_CROSSHAIR);

/* change the pointer to the application pointer */
WinSetPointer(HWND_DESKTOP, hptrCrossHair);

/* restore the system pointer */
WinSetPointer(HWND_DESKTOP, hptrDefault);

```

WinQuerySysPointer - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQuerySysPointerData

WinQuerySysPointerData - Syntax

This function is specific to OS/2 Version 2.1 or higher.

This function returns the icon data for the specified system pointer for application use.

```

#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndDesktop; /* Handle to the desktop window. */
ULONG     iptr;         /* The index of the desired system pointer. */
PICONINFO pIconInfo;    /* Icon data buffer. */
BOOL      rc;           /* Return value. */

rc = WinQuerySysPointerData(hwndDesktop, iptr,
                             pIconInfo);

```

WinQuerySysPointerData Parameter - hwndDesktop

hwndDesktop ([HWND](#)) - input
Handle to the desktop window.

WinQuerySysPointerData Parameter - iptr

iptr ([ULONG](#)) - input
The index of the desired system pointer.

Possible values are in the following list:

SPTR_ARROW	Arrow pointer
SPTR_TEXT	Text I-beam pointer
SPTR_WAIT	Wait icon pointer
SPTR_MOVE	Move pointer
SPTR_SIZENWSE	Downward-sloping, double-headed arrow pointer
SPTR_SIZENESW	Upward-sloping, double-headed arrow pointer
SPTR_SIZEWE	Horizontal, double-headed arrow pointer
SPTR_SIZENS	Vertical, double-headed arrow pointer
SPTR_APPICON	Standard application icon pointer
SPTR_ICONINFORMATION	Information icon pointer
SPTR_ICONQUESICON	Question mark icon pointer
SPTR_ICONERROR	Exclamation mark icon pointer
SPTR_ICONWARNING	Warning icon pointer
SPTR_ILLEGAL	Illegal operation icon pointer
SPTR_FILE	Single file icon pointer
SPTR_MULTIFILE	Multiple files icon pointer
SPTR_FOLDER	Folder icon pointer
SPTR_PROGRAM	Application program icon pointer

WinQuerySysPointerData Parameter - plconInfo

plconInfo ([PICONINFO](#)) - in/out
Icon data buffer.

The pointer to the buffer the icon data is to be written into. The buffer should be large enough to accommodate the icon data.

WinQuerySysPointerData Return Value - rc

rc (**BOOL**) - returns
Return value.

TRUE	Successful.
FALSE	An error occurred.

WinQuerySysPointerData - Parameters

hwndDesktop (**HWND**) - input
Handle to the desktop window.

iptr (**ULONG**) - input
The index of the desired system pointer.

Possible values are in the following list:

SPTR_ARROW	Arrow pointer
SPTR_TEXT	Text I-beam pointer
SPTR_WAIT	Wait icon pointer
SPTR_MOVE	Move pointer
SPTR_SIZENWSE	Downward-sloping, double-headed arrow pointer
SPTR_SIZENESW	Upward-sloping, double-headed arrow pointer
SPTR_SIZEWE	Horizontal, double-headed arrow pointer
SPTR_SIZENS	Vertical, double-headed arrow pointer
SPTR_APPICON	Standard application icon pointer
SPTR_ICONINFORMATION	Information icon pointer
SPTR_ICONQUESICON	Question mark icon pointer
SPTR_ICONERROR	Exclamation mark icon pointer
SPTR_ICONWARNING	Warning icon pointer
SPTR_ILLEGAL	Illegal operation icon pointer
SPTR_FILE	Single file icon pointer
SPTR_MULTIFILE	Multiple files icon pointer
SPTR_FOLDER	Folder icon pointer
SPTR_PROGRAM	Application program icon pointer

plconInfo (**PICONINFO**) - in/out
Icon data buffer.

The pointer to the buffer the icon data is to be written into. The buffer should be large enough to accommodate the icon data.

rc ([BOOL](#)) - returns
Return value.

TRUE

Successful.

FALSE

An error occurred.

WinQuerySysPointerData - Remarks

The icon data is always returned in `ICON_DATA` format. The buffer supplied in `IconInfo` must be large enough to accommodate the icon data. If the buffer size provided is 0 or not large enough, the actual buffer size needed is returned in the [ICONINFO](#) structure.

WinQuerySysPointerData - Related Functions

Related Functions

- [WinCreatePointer](#)
- [WinCreatePointerIndirect](#)
- [WinDestroyPointer](#)
- [WinDrawPointer](#)
- [WinLoadPointer](#)
- [WinQueryPointer](#)
- [WinQueryPointerInfo](#)
- [WinQueryPointerPos](#)
- [WinQuerySysPointer](#)
- [WinQuerySysPointerData](#)
- [WinSetPointer](#)
- [WinSetPointerPos](#)
- [WinSetSysPointerData](#)
- [WinShowPointer](#)

WinQuerySysPointerData - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Related Functions](#)

[Glossary](#)

WinQuerySystemAtomTable

WinQuerySystemAtomTable - Syntax

This function returns the handle of the system atom table.

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HATOMTBL    hatomtblAtomTbl; /* System atom-table handle. */

hatomtblAtomTbl = WinQuerySystemAtomTable();
```

WinQuerySystemAtomTable Return Value - hatomtblAtomTbl

hatomtblAtomTbl ([HATOMTBL](#)) - returns
System atom-table handle.

WinQuerySystemAtomTable - Parameters

hatomtblAtomTbl ([HATOMTBL](#)) - returns
System atom-table handle.

WinQuerySystemAtomTable - Remarks

The system atom table can be accessed by any process in the system. It is created at boot time and cannot be destroyed.

WinQuerySystemAtomTable - Related Functions

Related Functions

- [WinAddAtom](#)
- [WinCreateAtomTable](#)
- [WinDeleteAtom](#)
- [WinDestroyAtomTable](#)
- [WinFindAtom](#)
- [WinQueryAtomLength](#)
- [WinQueryAtomName](#)
- [WinQueryAtomUsage](#)
- [WinQuerySystemAtomTable](#)

WinQuerySystemAtomTable - Example Code

This function queries the length of an atom.

```
#define INCL_WINATOM
#include <OS2.H>

HATOMTBL      hatomtbl;
ATOM          atom;
unsigned char  szAtomName;

hatomtbl = WinQuerySystemAtomTable();

atom = WinFindAtom(hatomtbl, &szAtomName);
```

WinQuerySystemAtomTable - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQuerySysValue

WinQuerySysValue - Syntax

This function returns a system value.

```
#define INCL_WINSYS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndDesktop; /* Desktop-window handle. */
LONG      iSysValue;    /* System-value identity. */
LONG      lValue;       /* System value. */

lValue = WinQuerySysValue(hwndDesktop, iSysValue);
```

WinQuerySysValue Parameter - hwndDesktop

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP
Return the system values for the desktop-window handle.

Other
Return the system values for the specified desktop-window handle.

WinQuerySysValue Parameter - iSysValue

iSysValue ([LONG](#)) - input
System-value identity.

This must be one of the following SV_* constants.

Note: Not all system values can be set with [WinSetSysValue](#); those that can be set are marked with an asterisk (*).

SV_ALARM
(*) TRUE if the alarm sound generated by [WinAlarm](#) is enabled; FALSE if the alarm sound is disabled.

SV_ALTMNEMONIC
(*) TRUE if the mnemonic is made up of KATAKANA characters; FALSE if the mnemonic is made up of ROMAN characters.

SV_ANIMATION
(*) TRUE when animation is set on. FALSE when animation is set off.

SV_BEGINDRAG
(*) Mouse begin drag (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_BEGINDRAGKB
(*) Keyboard begin drag (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_BEGINSELECT
(*) Mouse begin swipe select (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_BEGINSELECTKB
(*) Keyboard begin swipe select (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_CICONTEXTLINES
(*) Maximum number of lines that the icon text may occupy for a minimized window.

SV_CONTEXTHELP
(*) Mouse control for pop-up menu (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_CONTEXTHELPKB
(*) Keyboard control for pop-up menu (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_CONTEXTMENU
(*) Mouse request pop-up menu (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_CONTEXTMENUKB
(*) Keyboard request pop-up menu (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_CMOUSEBUTTONS
The number of buttons on the pointing device (zero if no pointing device is installed).

SV_CTIMERS
Count of available timers.

SV_CURSORLEVEL
The cursor hide level.

SV_CURSORRATE
(*) Cursor blink rate, in milliseconds.

SV_CXBORDER
Width of the nominal-width border.

SV_CXBYTEALIGN
Horizontal count of pels for alignment.

SV_CXDBLCLK
(*) Width of the pointing device double-click sensitive area. The default is the system-font character width.

SV_CXDLGFRAME
Width of the dialog-frame border.

SV_CXFULLSCREEN
Width of the client area when the window is full screen.

SV_CXHSCROLLARROW
Width of the horizontal scroll-bar arrow bit maps.

SV_CXHSLIDER
Width of the horizontal scroll-bar thumb.

SV_CXICON
Icon width.

SV_CXICONTEXTWIDTH
(*) Maximum number of characters per line allowed in the icon text for a minimized window.

SV_CXMINMAXBUTTON
Width of the minimize/maximize buttons.

SV_CXMOTIONSTART
(*) The number of pels that a pointing device must be moved in the horizontal direction, while the button is depressed, before a WM_BUTTONxMOTIONSTR message is sent.

SV_CXPOINTER
Pointer width.

SV_CXSCREEN
Width of the screen.

SV_CXSIZEBORDER
(*) Width of the sizing border.

SV_CXVSCROLL
Width of the vertical scroll-bar.

SV_CYBORDER
Height of the nominal-width border.

SV_CYBYTEALIGN
Vertical count of pels for alignment.

SV_CYDBLCLK
(*) Height of the pointing device double-click sensitive area. The default is half the height of the system font character height.

SV_CYDLGFRAME
Height of the dialog-frame border.

SV_CYFULLSCREEN
Height of the client area when the window is full screen (excluding menu height).

SV_CYHSCROLL
Height of the horizontal scroll-bar.

SV_CYICON
Icon height.

SV_CYMENU
Height of the single-line menu height.

SV_CYMINMAXBUTTON
Height of the minimize/maximize buttons.

SV_CYMOTIONSTART
(*) The number of pels that a pointing device must be moved in the vertical direction, while the button is depressed, before a WM_BUTTONxMOTIONSTR message is sent.

SV_CYPOINTER
Pointer height.

SV_CYSCREEN
Height of the screen.

SV_CYSIZEBORDER
(*) Height of the sizing border.

SV_CYTITLEBAR
Height of the caption.

SV_CYVSCROLLARROW
Height of the vertical scroll-bar arrow bit maps.

SV_CYVSLIDER
Height of the vertical scroll-bar thumb.

SV_DBLCLKTIME
(*) Pointing device double-click time, in milliseconds.

SV_DEBUG
FALSE indicates this is not a debug system.

SV_ENDDRAG
(*) Mouse end drag (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_ENDDRAGKB
(*) Keyboard end drag (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_ENDSELECT
(*) Mouse select or end swipe select (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_ENDSELECTKB
(*) Keyboard select or end swipe select (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_ERRORDURATION
(*) Duration for error alarms generated by [WinAlarm](#).

SV_ERRORFREQ
(*) Frequency for error alarms generated by [WinAlarm](#).

SV_EXTRAKEYBEEP
(*) When TRUE, the press of a key that does not exist on the Enhanced keyboard causes the system to generate a beep.

SV_FIRSTSCROLLRATE
(*) The delay (in milliseconds) before autoscrolling starts, when using a scroll bar.

SV_INSERTMODE
(*) TRUE if the system is in insert mode (for edit and multi-line edit controls); FALSE if in overwrite mode.

This system value is toggled by the system when the insert key is toggled, regardless of which window has the focus at the time.

SV_KBDALTERED
(*) Hardware ID of the newly attached keyboard.

Note: The OS/2 National Language Support is only loaded once per system IPL. The OS/2 NLS translation is based partially on the type of keyboard device attached to the system. There are two main keyboard device types:

PC AT styled and Enhanced styled. Hot Plugging between these two types of devices may result in typing anomalies due to a mismatch in the NLS device tables loaded and that of the attached device. It is strongly recommended that keyboard hot plugging be limited to the device type that the system was IPL'd with. In addition, OS/2 support will default to the 101/102 key Enhanced keyboard if no keyboard or a NetServer Mode password was in use during system IPL. (See Category 4, IOCTls 77h and 7Ah for more information on keyboard devices and types.)

SV_LOCKSTARTINPUT

(*) TRUE when the type ahead function is enabled; FALSE when the type ahead function is disabled.

SV_MENUROLLODOWNDELAY

(*) The delay in milliseconds before displaying a pull down referred to from a submenu item, when the button is already down as the pointer moves onto the submenu item.

SV_MENUROLLUPDELAY

(*) The delay in milliseconds before hiding a pull down referred to from a submenu item, when the button is already down as the pointer moves off the submenu item.

SV_MONOICONS

(*) When TRUE preference is given to black and white icons when selecting which icon resource definition to use on the screen. Black and white icons may have more clarity than color icons on LCD and Plasma display screens.

SV_MOUSEPRESENT

When TRUE a mouse pointing device is attached to the system.

SV_NOTEDURATION

(*) Duration for note alarms generated by [WinAlarm](#).

SV_NOTEFREQ

(*) Frequency for note alarms generated by [WinAlarm](#).

SV_OPEN

(*) Mouse open (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_OPENKB

(*) Keyboard open (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_POINTERLEVEL

Pointer hide level. If the pointer level is zero, the pointer is visible. If it is greater than zero, the pointer is not visible. The [WinShowPointer](#) call is invoked to increment and decrement the SV_POINTERLEVEL, but its value cannot become negative.

SV_PRINTSCREEN

(*) TRUE when the Print Screen function is enabled; FALSE when the Print Screen function is disabled.

SV_SCROLLRATE

(*) The delay (in milliseconds) between scroll operations, when using a scroll bar.

SV_SETLIGHTS

(*) When TRUE, the appropriate light is set when the keyboard state table is set.

SV_SINGLESELECT

(*) Mouse select (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_TASKLISTMOUSEACCESS

(*) Determines whether the task list is displayed when mouse buttons 1 and 2 are pressed simultaneously, or when mouse button 2 is pressed by itself, or for no mouse gesture.

SV_TEXTEDIT

(*) Mouse begin direct name edit (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_TEXTEDITKB

(*) Keyboard begin direct name edit (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_TRACKRECTLEVEL

The hide level of the tracking rectangle (zero if visible, greater than zero if not).

SV_SWAPBUTTON

(*) TRUE if pointing device buttons are swapped. Normally, the pointing device buttons are set for right-handed use. Setting this value changes them for left-handed use.

If TRUE, WM_LBUTTONDOWN* messages are returned when the user presses the right button, and WM_RBUTTONDOWN* messages are returned when the left button is pressed. Modifying this value affects the entire system. Applications

should not normally read or set this value; users update this value by means of the user interface shell to suit their requirements.

SV_WARNINGDURATION

(*) Duration for warning alarms generated by [WinAlarm](#).

SV_WARNINGFREQ

(*) Frequency for warning alarms generated by [WinAlarm](#).

WinQuerySysValue Return Value - IValue

IValue ([LONG](#)) - returns

System value.

0

Error occurred

Other

System value. Dimensions are in pels and times are in milliseconds.

WinQuerySysValue - Parameters

hwndDeskTop ([HWND](#)) - input

Desktop-window handle.

HWND_DESKTOP

Return the system values for the desktop-window handle.

Other

Return the system values for the specified desktop-window handle.

iSysValue ([LONG](#)) - input

System-value identity.

This must be one of the following SV_* constants.

Note: Not all system values can be set with [WinSetSysValue](#); those that can be set are marked with an asterisk (*).

SV_ALARM

(*) TRUE if the alarm sound generated by [WinAlarm](#) is enabled; FALSE if the alarm sound is disabled.

SV_ALTMNEMONIC

(*) TRUE if the mnemonic is made up of KATAKANA characters; FALSE if the mnemonic is made up of ROMAN characters.

SV_ANIMATION

(*) TRUE when animation is set on. FALSE when animation is set off.

SV_BEGINDRAG

(*) Mouse begin drag (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_BEGINDRAGKB

(*) Keyboard begin drag (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_BEGINSELECT

(*) Mouse begin swipe select (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_BEGINSELECTKB

(*) Keyboard begin swipe select (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_CICONTEXTLINES
(*) Maximum number of lines that the icon text may occupy for a minimized window.

SV_CONTEXTHELP
(*) Mouse control for pop-up menu (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_CONTEXTHELPKB
(*) Keyboard control for pop-up menu (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_CONTEXTMENU
(*) Mouse request pop-up menu (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_CONTEXTMENUKB
(*) Keyboard request pop-up menu (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_CMOUSEBUTTONS
The number of buttons on the pointing device (zero if no pointing device is installed).

SV_CTIMERS
Count of available timers.

SV_CURSORLEVEL
The cursor hide level.

SV_CURSORRATE
(*) Cursor blink rate, in milliseconds.

SV_CXBORDER
Width of the nominal-width border.

SV_CXBYTEALIGN
Horizontal count of pels for alignment.

SV_CXDBLCLK
(*) Width of the pointing device double-click sensitive area. The default is the system-font character width.

SV_CXDLGFRAME
Width of the dialog-frame border.

SV_CXFULLSCREEN
Width of the client area when the window is full screen.

SV_CXHSCROLLARROW
Width of the horizontal scroll-bar arrow bit maps.

SV_CXHSLIDER
Width of the horizontal scroll-bar thumb.

SV_CXICON
Icon width.

SV_CXICONTEXTWIDTH
(*) Maximum number of characters per line allowed in the icon text for a minimized window.

SV_CXMINMAXBUTTON
Width of the minimize/maximize buttons.

SV_CXMOTIONSTART
(*) The number of pels that a pointing device must be moved in the horizontal direction, while the button is depressed, before a WM_BUTTONxMOTIONSTR message is sent.

SV_CXPOINTER
Pointer width.

SV_CXSCREEN
Width of the screen.

SV_CXSIZEBORDER
(*) Width of the sizing border.

SV_CXVSCROLL

Width of the vertical scroll-bar.

SV_CYBORDER
Height of the nominal-width border.

SV_CYBYTEALIGN
Vertical count of pels for alignment.

SV_CYDBLCLK
(*) Height of the pointing device double-click sensitive area. The default is half the height of the system font character height.

SV_CYDLGFRAME
Height of the dialog-frame border.

SV_CYFULLSCREEN
Height of the client area when the window is full screen (excluding menu height).

SV_CYHSCROLL
Height of the horizontal scroll-bar.

SV_CYICON
Icon height.

SV_CYMENU
Height of the single-line menu height.

SV_CYMINMAXBUTTON
Height of the minimize/maximize buttons.

SV_CYMOTIONSTART
(*) The number of pels that a pointing device must be moved in the vertical direction, while the button is depressed, before a WM_BUTTONxMOTIONSTR message is sent.

SV_CYPOINTER
Pointer height.

SV_CYSCREEN
Height of the screen.

SV_CYSIZEBORDER
(*) Height of the sizing border.

SV_CYTITLEBAR
Height of the caption.

SV_CYVSCROLLARROW
Height of the vertical scroll-bar arrow bit maps.

SV_CYVSLIDER
Height of the vertical scroll-bar thumb.

SV_DBLCLKTIME
(*) Pointing device double-click time, in milliseconds.

SV_DEBUG
FALSE indicates this is not a debug system.

SV_ENDDRAG
(*) Mouse end drag (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_ENDDRAGKB
(*) Keyboard end drag (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_ENDSELECT
(*) Mouse select or end swipe select (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_ENDSELECTKB
(*) Keyboard select or end swipe select (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_ERRORDURATION

(*) Duration for error alarms generated by [WinAlarm](#).

SV_ERRORFREQ

(*) Frequency for error alarms generated by [WinAlarm](#).

SV_EXTRAKEYBEEP

(*) When TRUE, the press of a key that does not exist on the Enhanced keyboard causes the system to generate a beep.

SV_FIRSTSCROLLRATE

(*) The delay (in milliseconds) before autoscrolling starts, when using a scroll bar.

SV_INSERTMODE

(*) TRUE if the system is in insert mode (for edit and multi-line edit controls); FALSE if in overtype mode.

This system value is toggled by the system when the insert key is toggled, regardless of which window has the focus at the time.

SV_KBDALTERED

(*) Hardware ID of the newly attached keyboard.

Note: The OS/2 National Language Support is only loaded once per system IPL. The OS/2 NLS translation is based partially on the type of keyboard device attached to the system. There are two main keyboard device types: PC AT styled and Enhanced styled. Hot Plugging between these two types of devices may result in typing anomalies due to a mismatch in the NLS device tables loaded and that of the attached device. It is strongly recommended that keyboard hot plugging be limited to the device type that the system was IPL'd with. In addition, OS/2 support will default to the 101/102 key Enhanced keyboard if no keyboard or a NetServer Mode password was in use during system IPL. (See Category 4, IOCTls 77h and 7Ah for more information on keyboard devices and types.)

SV_LOCKSTARTINPUT

(*) TRUE when the type ahead function is enabled; FALSE when the type ahead function is disabled.

SV_MENUROLLDOWNDELAY

(*) The delay in milliseconds before displaying a pull down referred to from a submenu item, when the button is already down as the pointer moves onto the submenu item.

SV_MENUROLLUPDELAY

(*) The delay in milliseconds before hiding a pull down referred to from a submenu item, when the button is already down as the pointer moves off the submenu item.

SV_MONOICONS

(*) When TRUE preference is given to black and white icons when selecting which icon resource definition to use on the screen. Black and white icons may have more clarity than color icons on LCD and Plasma display screens.

SV_MOUSEPRESENT

When TRUE a mouse pointing device is attached to the system.

SV_NOTEDURATION

(*) Duration for note alarms generated by [WinAlarm](#).

SV_NOTEFREQ

(*) Frequency for note alarms generated by [WinAlarm](#).

SV_OPEN

(*) Mouse open (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_OPENKB

(*) Keyboard open (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_POINTERLEVEL

Pointer hide level. If the pointer level is zero, the pointer is visible. If it is greater than zero, the pointer is not visible. The [WinShowPointer](#) call is invoked to increment and decrement the SV_POINTERLEVEL, but its value cannot become negative.

SV_PRINTSCREEN

(*) TRUE when the Print Screen function is enabled; FALSE when the Print Screen function is disabled.

SV_SCROLLRATE

(*) The delay (in milliseconds) between scroll operations, when using a scroll bar.

SV_SETLIGHTS

(*) When TRUE, the appropriate light is set when the keyboard state table is set.

SV_SINGLESELECT
 (*) Mouse select (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_TASKLISTMOUSEACCESS
 (*) Determines whether the task list is displayed when mouse buttons 1 and 2 are pressed simultaneously, or when mouse button 2 is pressed by itself, or for no mouse gesture.

SV_TEXTEDIT
 (*) Mouse begin direct name edit (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_TEXTEDITKB
 (*) Keyboard begin direct name edit (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_TRACKRECTLEVEL
 The hide level of the tracking rectangle (zero if visible, greater than zero if not).

SV_SWAPBUTTON
 (*) TRUE if pointing device buttons are swapped. Normally, the pointing device buttons are set for right-handed use. Setting this value changes them for left-handed use.

If TRUE, WM_LBUTTONDOWN* messages are returned when the user presses the right button, and WM_RBUTTONDOWN* messages are returned when the left button is pressed. Modifying this value affects the entire system. Applications should not normally read or set this value; users update this value by means of the user interface shell to suit their requirements.

SV_WARNINGDURATION
 (*) Duration for warning alarms generated by [WinAlarm](#).

SV_WARNINGFREQ
 (*) Frequency for warning alarms generated by [WinAlarm](#).

IValue ([LONG](#)) - returns
 System value.

0
 Error occurred

Other
 System value. Dimensions are in pels and times are in milliseconds.

WinQuerySysValue - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
 An invalid window handle was specified.

PMERR_PARAMETER_OUT_OF_RANGE (0x1003)
 The value of a parameter was not within the defined valid range for that parameter.

WinQuerySysValue - Related Functions

Related Functions

- [WinQuerySysValue](#)
- [WinSetSysValue](#)

WinQuerySysValue - Example Code

This example uses the WinQuerySysValue function to query the sizing border dimensions.

```
#define INCL_WINSYS
#include <OS2.H>

LONG vlXBorder, vlYBorder;

vlXBorder = WinQuerySysValue(HWND_DESKTOP,
                             SV_CXSIZEBORDER);
vlYBorder = WinQuerySysValue(HWND_DESKTOP,
                             SV_CYSIZEBORDER);
```

WinQuerySysValue - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryTaskSizePos

WinQueryTaskSizePos - Syntax

This function obtains the recommended size, position and status for the first window of a newly started application (typically the main window).

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab; /* Anchor-block handle. */
ULONG    ulID; /* Session. */
PSWP     pswp; /* Window position and size data. */
ULONG    rc; /* Return code. */

rc = WinQueryTaskSizePos(hab, ulID, pswp);
```

WinQueryTaskSizePos Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinQueryTaskSizePos Parameter - ulID

ulID ([ULONG](#)) - input
Session.

If zero is specified, the session number of the caller is used.

WinQueryTaskSizePos Parameter - pswp

pswp ([PSWP](#)) - output
Window position and size data.

Contains the recommended size and position for the first (main) window of the application. The window flags are set to indicate whether this window should be activated, minimized, or maximized.

WinQueryTaskSizePos Return Value - rc

rc ([ULONG](#)) - returns
Return code.

0	Successful completion
Other	Error occurred.

WinQueryTaskSizePos - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

ulID ([ULONG](#)) - input
Session.

If zero is specified, the session number of the caller is used.

pswp ([PSWP](#)) - output
Window position and size data.


```
WinSetWindowPos(hwndFrame, HWND_TOP,
    winpos.x,          /* x pos */
    winpos.y,          /* y pos */
    winpos.cx,         /* x size */
    winpos.cy,         /* y size */
    SWP_ACTIVATE | SWP_MOVE | SWP_SIZE | SWP_SHOW); /* flags*/
```

WinQueryTaskSizePos - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinQueryTaskTitle

WinQueryTaskTitle - Syntax

This function obtains the title under which a specified application is started, or is added to the Window List. (See also [WinQuerySessionTitle](#), which you should use for preference.)

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

ULONG    ulSession;    /* Session identity of application whose title is requested. */
PSZ      pszTitle;     /* Window List title. */
ULONG    ulTitlelen;   /* Maximum length of data returnable, in bytes. */
ULONG    rc;           /* Return code. */

rc = WinQueryTaskTitle(ulSession, pszTitle,
    ulTitlelen);
```

WinQueryTaskTitle Parameter - ulSession

ulSession ([ULONG](#)) - input

Session identity of application whose title is requested.

0

Use the session identity of the caller

Other

Use the specified session identity.

WinQueryTaskTitle Parameter - pszTitle

pszTitle ([PSZ](#)) - output
Window List title.

This is the title of the application with a process identity, if the application is present in the Window List.

WinQueryTaskTitle Parameter - ulTitlelen

ulTitlelen ([ULONG](#)) - input
Maximum length of data returnable, in bytes.

If the *pszTitle* parameter is longer than this, the title is truncated. However, the terminating null character is left at the end of the string. The maximum number of title characters copied is (*ulTitlelen*-1).

WinQueryTaskTitle Return Value - rc

rc ([ULONG](#)) - returns
Return code.

0	Successful completion
Other	Error occurred.

WinQueryTaskTitle - Parameters

ulSession ([ULONG](#)) - input
Session identity of application whose title is requested.

0	Use the session identity of the caller
Other	Use the specified session identity.

pszTitle ([PSZ](#)) - output
Window List title.

This is the title of the application with a process identity, if the application is present in the Window List.

ulTitlelen ([ULONG](#)) - input
Maximum length of data returnable, in bytes.

If the *pszTitle* parameter is longer than this, the title is truncated. However, the terminating null character is left at the end of the string. The maximum number of title characters copied is (*ulTitleLen*-1).

rc ([ULONG](#)) - returns
Return code.

0	Successful completion
Other	Error occurred.

WinQueryTaskTitle - Remarks

This function is useful when an application uses the same name in its window title (and in its entry in the Window List) as the end user invokes to start the application. This provides a visual link for the end user.

If this function is used after a Window List entry is created for the application, the title in the Window List entry is obtained.

WinQueryTaskTitle - Related Functions

Related Functions

- [WinAddSwitchEntry](#)
- [WinChangeSwitchEntry](#)
- [WinCreateSwitchEntry](#)
- [WinQuerySessionTitle](#)
- [WinQuerySwitchEntry](#)
- [WinQuerySwitchHandle](#)
- [WinQuerySwitchList](#)
- [WinQueryTaskSizePos](#)
- [WinQueryTaskTitle](#)
- [WinRemoveSwitchEntry](#)
- [WinSwitchToProgram](#)

WinQueryTaskTitle - Example Code

This example calls WinQueryTaskTitle to retrieve the application's title, and then sets the title bar of the frame window to that title. (The WinQuerySessionTitle could be used instead).

```
#define INCL_WINSWITCHLIST
#include <OS2.H>

HAB      hab;
HWND     hwndFrame, hwndClient;
CHAR     szTitle[MAXNAMEL + 1];
HSWITCH  hswitch;
SWCNTRL  swctl;

hswitch = WinQuerySwitchHandle(hwndFrame, 0);
WinQuerySwitchEntry(hswitch, &swctl);

WinQueryTaskTitle(0,
                  szTitle,
                  sizeof(szTitle));
```

```
hwndFrame = WinQueryWindow(hwndClient,
                           QW_PARENT); /* get handle of parent, */
                                       /* which is frame window. */
WinSetWindowText(hwndFrame, szTitle);
```

WinQueryTaskTitle - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinQueryUpdateRect

WinQueryUpdateRect - Syntax

This function returns the rectangle that bounds the update region of a specified window.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwnd;      /* Handle of window whose update rectangle is to be queried. */
PRECTL    prclPrc;    /* Update region that bounds the rectangle (in window coordinates). */
BOOL      rc;         /* Success indicator. */

rc = WinQueryUpdateRect(hwnd, prclPrc);
```

WinQueryUpdateRect Parameter - hwnd

hwnd (**HWND**) - input

Handle of window whose update rectangle is to be queried.

WinQueryUpdateRect Parameter - prclPrc

prcIPrc ([PRECTL](#)) - output
Update region that bounds the rectangle (in window coordinates).

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) may also be used, if supported by the language.

WinQueryUpdateRect Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred, or window has no update region; it is wholly valid, therefore <i>prcIPrc</i> is NULL.

WinQueryUpdateRect - Parameters

hwnd ([HWND](#)) - input
Handle of window whose update rectangle is to be queried.

prcIPrc ([PRECTL](#)) - output
Update region that bounds the rectangle (in window coordinates).

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) may also be used, if supported by the language.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred, or window has no update region; it is wholly valid, therefore <i>prcIPrc</i> is NULL.

WinQueryUpdateRect - Remarks

This function is useful for implementing an incremental update scheme as an alternative to the [WinBeginPaint](#) and [WinEndPaint](#) functions.

WinQueryUpdateRect - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinQueryUpdateRect - Related Functions

Related Functions

- [WinBeginPaint](#)
 - [WinEnableWindowUpdate](#)
 - [WinEndPaint](#)
 - [WinExcludeUpdateRegion](#)
 - [WinGetClipPS](#)
 - [WinGetPS](#)
 - [WinGetScreenPS](#)
 - [WinInvalidateRect](#)
 - [WinInvalidateRegion](#)
 - [WinIsWindowShowing](#)
 - [WinIsWindowVisible](#)
 - [WinLockVisRegions](#)
 - [WinOpenWindowDC](#)
 - [WinQueryUpdateRect](#)
 - [WinQueryUpdateRegion](#)
 - [WinRealizePalette](#)
 - [WinReleasePS](#)
 - [WinShowWindow](#)
 - [WinUpdateWindow](#)
 - [WinValidateRect](#)
 - [WinValidateRegion](#)
-

WinQueryUpdateRect - Example Code

This example assumes that `WinInvalidateRect` has been called on a window owned by this window procedure. The application is sent the `WM_PAINT` message with the entire window as the update rectangle. The window procedure gets the dimensions of the window and calls `WinValidateRect` to revalidate the window.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HWND  hwnd;
RECT  rcl;
.
.
.
CASE WM_PAINT:

    WinQueryUpdateRect(hwnd, &rcl);
    WinValidateRect(hwnd, &rcl, FALSE);
    WinFillRect(hwnd, &rcl, CLR_RED);
    return 0L;
```

WinQueryUpdateRect - Topics

Select an item:
[Syntax](#)

[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryUpdateRegion

WinQueryUpdateRegion - Syntax

This call obtains an update region of a window.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd;           /* Handle of window whose update region is to be queried. */
HRGN    hrgn;           /* Handle of the window's update region. */
LONG    lComplexity;    /* Complexity of resulting region/error indicator. */

lComplexity = WinQueryUpdateRegion(hwnd, hrgn);
```

WinQueryUpdateRegion Parameter - hwnd

hwnd ([HWND](#)) - input
Handle of window whose update region is to be queried.

WinQueryUpdateRegion Parameter - hrgn

hrgn ([HRGN](#)) - input
Handle of the window's update region.

The window's update region, in window coordinates, is copied into *hrgn*.

WinQueryUpdateRegion Return Value - lComplexity

lComplexity ([LONG](#)) - returns
Complexity of resulting region/error indicator.

RGN_NULL	Null region
RGN_RECT	Rectangular region
RGN_COMPLEX	Complex region
RGN_ERROR	Error.

WinQueryUpdateRegion - Parameters

hwnd ([HWND](#)) - input
Handle of window whose update region is to be queried.

hrgn ([HRGN](#)) - input
Handle of the window's update region.

The window's update region, in window coordinates, is copied into *hrgn*.

lComplexity ([LONG](#)) - returns
Complexity of resulting region/error indicator.

RGN_NULL	Null region
RGN_RECT	Rectangular region
RGN_COMPLEX	Complex region
RGN_ERROR	Error.

WinQueryUpdateRegion - Remarks

This call is useful for implementing an alternate update scheme to those used by the [WinBeginPaint](#), and [WinEndPaint](#) functions, together with the [WinValidateRegion](#) function.

The application can use the returned update region as the clip region for a presentation space, so that drawing output can be clipped to the window's update region.

WinQueryUpdateRegion - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_HRGN_BUSY (0x2034)
An internal region busy error was detected. The region was locked by one thread during an attempt to access it from another thread.

WinQueryUpdateRegion - Related Functions

Related Functions

- [WinBeginPaint](#)
- [WinEnableWindowUpdate](#)
- [WinEndPaint](#)
- [WinExcludeUpdateRegion](#)
- [WinGetClipPS](#)
- [WinGetPS](#)
- [WinGetScreenPS](#)
- [WinInvalidateRect](#)
- [WinInvalidateRegion](#)
- [WinIsWindowShowing](#)
- [WinIsWindowVisible](#)
- [WinLockVisRegions](#)
- [WinOpenWindowDC](#)
- [WinQueryUpdateRect](#)
- [WinQueryUpdateRegion](#)
- [WinRealizePalette](#)
- [WinReleasePS](#)
- [WinShowWindow](#)
- [WinUpdateWindow](#)
- [WinValidateRect](#)
- [WinValidateRegion](#)

WinQueryUpdateRegion - Example Code

This example gets the region that needs to be updated and then repaints the invalid region, if necessary.

```
#define INCL_WINWINDOWMGR
#define INCL_GPIREGIONS
#include <OS2.H>

HWND hwnd;
HRGN hrgn; /* region handle. */

if (RGN_NULL != WinQueryUpdateRegion(hwnd, hrgn)) {
    /* repaint the invalid region */
    .
    .
    .
}
```

WinQueryUpdateRegion - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryVersion

WinQueryVersion - Syntax

This function is included for compatibility purposes only, and its use is not recommended. It is recommended that DosQuerySysInfo (see the *Control Program Programming Reference*) be used to return the version, the revision level, and the environment of PM.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HAB      hab;      /* Anchor-block handle. */
ULONG    fSysInf;   /* System information within which the application is operating. */

fSysInf = WinQueryVersion(hab);
```

WinQueryVersion Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinQueryVersion Return Value - fSysInf

fSysInf (**ULONG**) - returns
System information within which the application is operating.

This information consists of 4 bytes packed into the ULONG variable as follows:

- The first byte, bits 0-7, contains the major version.
- The second byte, bits 8-15, contains the minor version.
- The third byte, bits 16-23, contains the revision number.
- The fourth byte, bits 24-31, is reserved.

Version	Revision	Minor	Major
OS/2 2.0	0	00	20
OS/2 2.1	0	10	20
OS/2 2.11	0	11	20
OS/2 Warp 3.0	0	30	20

WinQueryVersion - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

fSysInf ([ULONG](#)) - returns
System information within which the application is operating.

This information consists of 4 bytes packed into the ULONG variable as follows:

- The first byte, bits 0-7, contains the major version.
- The second byte, bits 8-15, contains the minor version.
- The third byte, bits 16-23, contains the revision number.
- The fourth byte, bits 24-31, is reserved.

Version	Revision	Minor	Major
OS/2 2.0	0	00	20
OS/2 2.1	0	10	20
OS/2 2.11	0	11	20
OS/2 Warp 3.0	0	30	20

WinQueryVersion - Example Code

This example gets the version of PM that is running.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
#include <stdio.h>

HAB hab;
ULONG lVer;

lVer = WinQueryVersion(hab);
printf("PM revision is %d ", lVer);
```

WinQueryVersion - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Example Code](#)

WinQueryVisibleRegion

WinQueryVisibleRegion - Syntax

This function is similar to [WinQueryUpdateRegion](#). It returns a region which could be NULL, rectangular, or complex, and represents the visible region of the window.

```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND     hwnd;    /* The window handle. */
HRGN     hrgn;    /* Region handle to receive the current visible region. */
ULONG    ulretn;

ulretn = WinQueryVisibleRegion(hwnd, hrgn);
```

WinQueryVisibleRegion Parameter - hwnd

hwnd ([HWND](#)) - input
The window handle.

WinQueryVisibleRegion Parameter - hrgn

hrgn ([HRGN](#)) - input
Region handle to receive the current visible region.

WinQueryVisibleRegion Return Value - ulretn

ulretn ([ULONG](#)) - returns

RGN_ERROR	An error occurred.
RGN_NULL	

RGN_RECT	The window currently has no visible area on screen.
RGN_COMPLEX	The region contains a series of rectangular areas.
	The region contains a complex shape.

WinQueryVisibleRegion - Parameters

hwnd ([HWND](#)) - input
The window handle.

hrgn ([HRGN](#)) - input
Region handle to receive the current visible region.

ulretn ([ULONG](#)) - returns

RGN_ERROR	An error occurred.
RGN_NULL	The window currently has no visible area on screen.
RGN_RECT	The region contains a series of rectangular areas.
RGN_COMPLEX	The region contains a complex shape.

WinQueryVisibleRegion - Remarks

This function returns the current visible region for the given window, even if that window does not receive notification when the visible region is changed. Currently, the region returned will always be a set of rectangular areas.

WinQueryVisibleRegion - Related Functions

Related Functions

- [WinQueryUpdateRegion](#)
 - [WinSetVisibleRegionNotify](#)
-

WinQueryVisibleRegion - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

WinQueryWindow

WinQueryWindow - Syntax

This function returns the handle of a window that has a specified relationship to a specified window.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd;           /* Handle of window to query. */
LONG    lCode;          /* Type of window information. */
HWND    hwndRelated;    /* Window handle. */

hwndRelated = WinQueryWindow(hwnd, lCode);
```

WinQueryWindow Parameter - hwnd

hwnd (**HWND**) - input
Handle of window to query.

WinQueryWindow Parameter - lCode

lCode (**LONG**) - input
Type of window information.

Determines what window information is returned:

QW_NEXT	Next window in z-order (window below).
QW_PREV	Previous window in z-order (window above).
QW_TOP	Topmost child window.
QW_BOTTOM	Bottommost child window.
QW_OWNER	Owner of window.
QW_PARENT	Parent of window.

QW_NEXTTOP

Returns the next window of the owner window hierarchy subject to their z-ordering.

The enumeration is evaluated in this order:

1. The hierarchy of windows owned by this window in their z-order.
2. The hierarchy of windows of the next z-ordered window having the same owner as this window.
3. The hierarchy of windows in their z-order having the same owner as the owner of this window. This step is repeated until the top of the owner tree for this window is reached.
4. The hierarchy of windows in their z-order of unowned windows.

QW_PREVTOP

Returns the previous main window, in the enumeration order defined by QW_NEXTTOP.

QW_FRAMEOWNER

Returns the owner of *hwnd* normalized so that it shares the same parent as *hwnd*.

WinQueryWindow Return Value - hwndRelated

hwndRelated (HWND) - returns

Window handle.

Handle of window related to *hwnd*.

WinQueryWindow - Parameters

hwnd (HWND) - input

Handle of window to query.

ICode (LONG) - input

Type of window information.

Determines what window information is returned:

QW_NEXT

Next window in z-order (window below).

QW_PREV

Previous window in z-order (window above).

QW_TOP

Topmost child window.

QW_BOTTOM

Bottommost child window.

QW_OWNER

Owner of window.

QW_PARENT

Parent of window.

QW_NEXTTOP

Returns the next window of the owner window hierarchy subject to their z-ordering.

The enumeration is evaluated in this order:

1. The hierarchy of windows owned by this window in their z-order.
2. The hierarchy of windows of the next z-ordered window having the same owner as this window.
3. The hierarchy of windows in their z-order having the same owner as the owner of this window. This step is repeated until the top of the owner tree for this window is reached.
4. The hierarchy of windows in their z-order of unowned windows.

QW_PREVTOP

Returns the previous main window, in the enumeration order defined by QW_NEXTTOP.

QW_FRAMEOWNER

Returns the owner of *hwnd* normalized so that it shares the same parent as *hwnd*.

hwndRelated (HWND) - returns
Window handle.

Handle of window related to *hwnd*.

WinQueryWindow - Remarks

If this function is used to enumerate windows of other threads, it cannot be ensured that all the windows are enumerated, because the z-ordering of the windows can change during the enumeration. [WinGetNextWindow](#) must be used for this purpose.

If this function is called with QW_OWNER or QW_PARENT, the return value is [WinQueryDesktopWindow](#)(hab, NULLHANDLE), and not HWND_DESKTOP, when the desktop window is reached.

If this function is called with QW_PARENT for an object window, the return value is the handle of the object window associated with the desktop window as returned by the [WinQueryObjectWindow](#) function.

WinQueryWindow - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_PARAMETER_OUT_OF_RANGE (0x1003)
The value of a parameter was not within the defined valid range for that parameter.

WinQueryWindow - Related Functions

Related Functions

- [WinBeginEnumWindows](#)
- [WinEndEnumWindows](#)
- [WinEnumDlgItem](#)
- [WinGetNextWindow](#)
- [WinIsChild](#)
- [WinMultWindowFromIDs](#)
- [WinQueryWindow](#)
- [WinSetOwner](#)
- [WinSetParent](#)

WinQueryWindow - Example Code

This example shows how to get the frame window handle from the client window handle.

```
#define INCL_WINWINDOWMGR
#define INCL_WINACCELERATORS
#include <OS2.H>

HACCEL haccel;
HWND hwndFrame, hwndClient; /* window handles. */
HAB hab; /* anchor block. */

hwndFrame = WinQueryWindow(hwndClient,
                           QW_PARENT); /* get handle of parent, */
                                       /* which is frame window. */
```

WinQueryWindow - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Errors](#)
- [Remarks](#)
- [Example Code](#)
- [Related Functions](#)
- [Glossary](#)

WinQueryWindowDC

WinQueryWindowDC - Syntax

This function returns the device context for a given window.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd; /* Window handle. */
HDC     hdc; /* Device-context handle. */

hdc = WinQueryWindowDC(hwnd);
```

WinQueryWindowDC Parameter - hwnd

hwnd ([HWND](#)) - input
Window handle.

WinQueryWindowDC Return Value - hdc

hdc ([HDC](#)) - returns
Device-context handle.

NULLHANDLE Either [WinOpenWindowDC](#) has not been called for this window, or an error has occurred.

Other Device context handle.

WinQueryWindowDC - Parameters

hwnd ([HWND](#)) - input
Window handle.

hdc ([HDC](#)) - returns
Device-context handle.

NULLHANDLE Either [WinOpenWindowDC](#) has not been called for this window, or an error has occurred.

Other Device context handle.

WinQueryWindowDC - Remarks

A handle is returned only if a device context has been opened for the window with [WinOpenWindowDC](#).

WinQueryWindowDC - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinQueryWindowDC - Related Functions

Related Functions

- [WinEnableWindow](#)
- [WinIsThreadActive](#)
- [WinIsWindow](#)
- [WinIsWindowEnabled](#)
- [WinQueryDesktopWindow](#)
- [WinQueryObjectWindow](#)
- [WinQueryWindowDC](#)
- [WinQueryWindowProcess](#)
- [WinQueryWindowRect](#)
- [WinWindowFromDC](#)
- [WinWindowFromID](#)
- [WinWindowFromPoint](#)

WinQueryWindowDC - Example Code

This example shows how to check if WinOpenWindowDC has been called for this window.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HWND  hwndClient;          /* window handle. */

if(WinQueryWindowDC(hwndClient))
{
    /* ... */
}
```

WinQueryWindowDC - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryWindowModel

WinQueryWindowModel - Syntax

This function queries the memory model associated with a window.

```
#define INCL_WINTHUNKAPI /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND     hwnd;      /* Window handle. */
ULONG     ulModel;   /* Memory model associated with the window. */

ulModel = WinQueryWindowModel(hwnd);
```

WinQueryWindowModel Parameter - hwnd

hwnd (**HWND**) - input
Window handle.

WinQueryWindowModel Return Value - ulModel

ulModel (**ULONG**) - returns
Memory model associated with the window.

PM_MODEL_1X	The 16-bit memory model of the 80386 processor.
PM_MODEL_2X	The 32-bit memory model of the 80386 processor.

WinQueryWindowModel - Parameters

hwnd (**HWND**) - input
Window handle.

ulModel (**ULONG**) - returns
Memory model associated with the window.

PM_MODEL_1X	The 16-bit memory model of the 80386 processor.
PM_MODEL_2X	The 32-bit memory model of the 80386 processor.

WinQueryWindowModel - Remarks

This function enables an application to query the memory model associated with a particular window to find out whether or not conversion of application-defined data is required. This may be necessary, for example, when sending DDE data. An existing OS/2 Version 1.1 or 1.2

application does not know about pointer conversion, so its data has to be converted for use in a 32-bit application.

The memory model is determined by how the window procedure was registered. If an application calls [WinRegisterClass](#) from 32-bit code, any windows created with that class are called 32-bit windows. If the application calls [WinSubclassWindow](#) from 16-bit code on a 32-bit window, that window becomes a 16-bit window.

WinQueryWindowModel - Related Functions

Related Functions

- [WinQueryClassThunkProc](#)
- [WinQueryWindowModel](#)
- [WinQueryWindowThunkProc](#)
- [WinSetClassThunkProc](#)
- [WinSetWindowThunkProc](#)

WinQueryWindowModel - Example Code

This example shows how to check if WinOpenWindowDC has been called for this window.

```
#define INCL_WINHOOKS
#define INCL_WINTHUNKAPI
#include <OS2.H>
HWND hwndClient;          /* window handle. */

if(WinQueryWindowModel(hwndClient) == PM_MODEL_2X)
{
    /* The 32-bit memory model of the 80386 processor. */
}
```

WinQueryWindowModel - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryWindowPos

WinQueryWindowPos - Syntax

This function queries the window size and position of a visible window.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd; /* Window handle. */
PSWP    pswp; /* SWP structure. */
BOOL    rc;    /* Success indicator. */

rc = WinQueryWindowPos(hwnd, pswp);
```

WinQueryWindowPos Parameter - hwnd

hwnd (**HWND**) - input
Window handle.

WinQueryWindowPos Parameter - pswp

pswp (**PSWP**) - output
SWP structure.

The fields are set such that a call to [WinSetWindowPos](#) with those values sets the window to its current size and position, with the exception of the // bits which are set as follows:

- SWP_MOVE and SWP_SIZE are set to TRUE.
- SWP_ACTIVATE and SWP_DEACTIVATE, are set to the current state of the window.
- If the window is minimized, SWP_MINIMIZE, is set and SWP_MAXIMIZE, is zero.
- If the window is maximized, SWP_MAXIMIZE, is set and SWP_MINIMIZE, is zero.
- If the window is neither minimized nor maximized, both SWP_MINIMIZE, and SWP_MAXIMIZE, are zero.
- All other bits are set to zero.

WinQueryWindowPos Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinQueryWindowPos - Parameters

hwnd ([HWND](#)) - input
Window handle.

pswp ([PSWP](#)) - output
SWP structure.

The fields are set such that a call to [WinSetWindowPos](#) with those values sets the window to its current size and position, with the exception of the // bits which are set as follows:

- SWP_MOVE and SWP_SIZE are set to TRUE.
- SWP_ACTIVATE and SWP_DEACTIVATE, are set to the current state of the window.
- If the window is minimized, SWP_MINIMIZE, is set and SWP_MAXIMIZE, is zero.
- If the window is maximized, SWP_MAXIMIZE, is set and SWP_MINIMIZE, is zero.
- If the window is neither minimized nor maximized, both SWP_MINIMIZE, and SWP_MAXIMIZE, are zero.
- All other bits are set to zero.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinQueryWindowPos - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_INVALID_FLAG (0x1019)
An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

WinQueryWindowPos - Related Functions

Related Functions

- [WinGetMinPosition](#)
- [WinQueryActiveWindow](#)
- [WinQueryWindowPos](#)
- [WinSaveWindowPos](#)
- [WinSetActiveWindow](#)
- [WinSetMultWindowPos](#)
- [WinSetWindowPos](#)

WinQueryWindowPos - Example Code

This example shows how to center a dialog box within the Screen using WinQueryWindowPos.

```
#define INCL_WINWINDOWMGR
```



```

#define INCL_WINSYS
#include <OS2.H>
HWND hwnd;          /* window handle. */
SHORT ix, iy;
SHORT iwidth, idepth;
SWP swp;

/* Query width and height of Screen device */
iwidth = WinQuerySysValue( HWND_DESKTOP, SV_CXSCREEN );
idepth = WinQuerySysValue( HWND_DESKTOP, SV_CYSCREEN );

/* Query width and height of dialog box */
WinQueryWindowPos( hwnd, (PSWP)&swp );

/* Center dialog box within the Screen */
ix = (SHORT)(( iwidth - swp.cx ) / 2);
iy = (SHORT)(( idepth - swp.cy ) / 2);
WinSetWindowPos( hwnd, HWND_TOP, ix, iy, 0, 0, SWP_MOVE );

```

WinQueryWindowPos - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryWindowProcess

WinQueryWindowProcess - Syntax

This function obtains the process identity and thread identity of the thread that created a window.

```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd; /* Window handle. */
PPID    ppid; /* Process identity of the thread that created the window. */
PTID    ptid; /* Thread identity of the thread that created the window. */
BOOL    rc;    /* Success indicator. */

rc = WinQueryWindowProcess(hwnd, ppid, ptid);

```

WinQueryWindowProcess Parameter - hwnd

hwnd (**HWND**) - input
Window handle.

WinQueryWindowProcess Parameter - ppid

ppid (**PPID**) - output
Process identity of the thread that created the window.

WinQueryWindowProcess Parameter - ptid

ptid (**PTID**) - output
Thread identity of the thread that created the window.

WinQueryWindowProcess Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinQueryWindowProcess - Parameters

hwnd (**HWND**) - input
Window handle.

ppid (**PPID**) - output
Process identity of the thread that created the window.

ptid (**PTID**) - output
Thread identity of the thread that created the window.

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinQueryWindowProcess - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinQueryWindowProcess - Related Functions

Related Functions

- [WinEnableWindow](#)
 - [WinIsThreadActive](#)
 - [WinIsWindow](#)
 - [WinIsWindowEnabled](#)
 - [WinQueryDesktopWindow](#)
 - [WinQueryObjectWindow](#)
 - [WinQueryWindowDC](#)
 - [WinQueryWindowProcess](#)
 - [WinQueryWindowRect](#)
 - [WinWindowFromDC](#)
 - [WinWindowFromID](#)
 - [WinWindowFromPoint](#)
-

WinQueryWindowProcess - Example Code

This example shows how to query a window's process and use that information to add a switch entry window.

```
#define INCL_WINWINDOWMGR
#define INCL_WINSYS
#include <OS2.H>

HWND      hwndFrame;          /* window handle. */
SWCNTRL   swctl;
PID       pid;
TID       tid;
HSWITCH   hsw;
char      szTitle[] = "app.exe";

WinQueryWindowProcess( hwndFrame, &pid, &tid);
swctl.hwnd = hwndFrame;
swctl.idProcess = pid;
strcpy( swctl.szSwtitle, szTitle);
hsw = WinAddSwitchEntry( &swctl);
```

WinQueryWindowProcess - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryWindowPtr

WinQueryWindowPtr - Syntax

This function retrieves a pointer value from the memory of the reserved window word.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd;    /* Window handle which has the pointer to retrieve. */
LONG    index;    /* Zero-based index of the pointer value to retrieve. */
PVOID    pRet;    /* Pointer value. */

pRet = WinQueryWindowPtr(hwnd, index);
```

WinQueryWindowPtr Parameter - hwnd

hwnd ([HWND](#)) - input
Window handle which has the pointer to retrieve.

WinQueryWindowPtr Parameter - index

index ([LONG](#)) - input
Zero-based index of the pointer value to retrieve.

The units of *index* are bytes. Valid values are zero through (*cbWindowData* -4), where *cbWindowData* is the parameter in [WinRegisterClass](#) that specifies the number of bytes available for application-defined storage.

The value QWP_PFNWP can be used for the address of the window's window procedure.

WinQueryWindowPtr Return Value - pRet

pRet ([PVOID](#)) - returns
Pointer value.

NULL	Error occurred.
Other	Pointer value.

WinQueryWindowPtr - Parameters

hwnd ([HWND](#)) - input
Window handle which has the pointer to retrieve.

index ([LONG](#)) - input
Zero-based index of the pointer value to retrieve.

The units of *index* are bytes. Valid values are zero through (*cbWindowData* -4), where *cbWindowData* is the parameter in [WinRegisterClass](#) that specifies the number of bytes available for application-defined storage.

The value QWP_PFNWP can be used for the address of the window's window procedure.

pRet ([PVOID](#)) - returns
Pointer value.

NULL	Error occurred.
Other	Pointer value.

WinQueryWindowPtr - Remarks

The *index* parameter is valid only if all of the bytes referenced are within the reserved memory.

WinQueryWindowPtr - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_PARAMETER_OUT_OF_RANGE (0x1003)
The value of a parameter was not within the defined valid range for that parameter.

WinQueryWindowPtr - Related Functions

Related Functions

- [WinQueryWindowPtr](#)
- [WinQueryWindowULong](#)
- [WinQueryWindowUShort](#)
- [WinSetWindowBits](#)
- [WinSetWindowPtr](#)
- [WinSetWindowULong](#)
- [WinSetWindowUShort](#)

WinQueryWindowPtr - Example Code

This function retrieves a pointer value from the memory of the reserved window word.

```
MyWindowProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
    MYINSTANCEDATA *InstanceData; /* application defined structure */

    switch (msg) {
    case WM_CREATE:
        DosAllocMem(&InstanceData, sizeof(MYINSTANCEDATA), fALLOC);
        /* WindowProcedure initializes instance data for this window */
        .
        .
        /* set pointer to instance in window words */
        WinSetWindowPtr(hwnd, 0, InstanceData);
        break;

    case WM_USER + 1: /* application defined message */
        /* Window procedure retrieves instance data to      */
        /* process this message                               */

        InstanceData = WinQueryWindowPtr(hwnd, 0);
        .
        .

        break;
        .
        .
    }
```

WinQueryWindowPtr - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryWindowRect

WinQueryWindowRect - Syntax

This function returns a window rectangle.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND     hwnd;          /* Window handle whose rectangle is retrieved. */
PRECTL   prclDest;      /* Window rectangle. */
BOOL     rc;            /* Rectangle-returned indicator. */

rc = WinQueryWindowRect(hwnd, prclDest);
```

WinQueryWindowRect Parameter - hwnd

hwnd (**HWND**) - input
Window handle whose rectangle is retrieved.

WinQueryWindowRect Parameter - prclDest

prclDest (**PRECTL**) - output
Window rectangle.

Window rectangle of *hwnd*, in window coordinates.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type **WRECT** may also be used, if supported by the language.

WinQueryWindowRect Return Value - rc

rc (**BOOL**) - returns
Rectangle-returned indicator.

TRUE	Rectangle successfully returned
FALSE	Rectangle not successfully returned.

WinQueryWindowRect - Parameters

hwnd ([HWND](#)) - input
Window handle whose rectangle is retrieved.

prclDest ([PRECTL](#)) - output
Window rectangle.

Window rectangle of *hwnd*, in window coordinates.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) may also be used, if supported by the language.

rc ([BOOL](#)) - returns
Rectangle-returned indicator.

TRUE	Rectangle successfully returned
FALSE	Rectangle not successfully returned.

WinQueryWindowRect - Remarks

The rectangle is in window coordinates relative to itself, so that the bottom left corner is at the position (0,0).

If the size of a frame window has been changed to zero by [WinSetWindowPos](#) or [WinSetMultWindowPos](#), the original size is returned because the window is hidden, not sized, in this instance.

WinQueryWindowRect - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinQueryWindowRect - Related Functions

- Related Functions**
- [WinEnableWindow](#)
 - [WinIsThreadActive](#)
 - [WinIsWindow](#)
 - [WinIsWindowEnabled](#)
 - [WinQueryDesktopWindow](#)
 - [WinQueryObjectWindow](#)
 - [WinQueryWindowDC](#)
 - [WinQueryWindowProcess](#)
 - [WinQueryWindowRect](#)
 - [WinWindowFromDC](#)
 - [WinWindowFromID](#)
 - [WinWindowFromPoint](#)

WinQueryWindowRect - Example Code

This example gets the dimensions of the window and calls WinInvalidateRect to invalidate the window. The application will be sent a WM_PAINT message with the entire window as the update rectangle.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HAB hab;
HWND hwnd;
RECTL rcl;

WinQueryWindowRect(hwnd, &rcl);
WinInvalidateRect(hwnd, /* window to invalidate */
                  &rcl, /* invalid rectangle */
                  FALSE); /* do not include children */
```

WinQueryWindowRect - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinQueryWindowText

WinQueryWindowText - Syntax

This function copies window text into a buffer.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND hwnd; /* Window handle. */
LONG lLength; /* Length of pun. */
PUN pun; /* Window text. */
LONG lRetLen; /* Length of returned text not including the null terminator. */

lRetLen = WinQueryWindowText(hwnd, lLength,
                             pun);
```

WinQueryWindowText Parameter - hwnd

hwnd (**HWND**) - input
Window handle.

If *hwnd* is a frame-window handle, the title-bar window text is copied.

WinQueryWindowText Parameter - ILength

ILength (**LONG**) - input
Length of *pun*.

It must be greater than 0.

WinQueryWindowText Parameter - pun

pun (**PUN**) - output
Window text.

WinQueryWindowText Return Value - IRetLen

IRetLen (**LONG**) - returns
Length of returned text not including the null terminator.

WinQueryWindowText - Parameters

hwnd (**HWND**) - input
Window handle.

If *hwnd* is a frame-window handle, the title-bar window text is copied.

ILength (**LONG**) - input
Length of *pun*.

It must be greater than 0.

pun (**PUN**) - output

Window text.

lRetLen ([LONG](#)) - returns

Length of returned text not including the null terminator.

WinQueryWindowText - Remarks

If the window text is longer than (*lLength*-1) only the first (*lLength*-1) characters of window text are copied.

If the window is the frame window, the title bar window text is copied.

This function sends a [WM_QUERYWINDOWPARAMS](#) message to *hwnd*.

If this function references the window of another process, *pwn* must be in memory that is shared by both processes, otherwise a memory fault can occur.

Note: Extreme caution must be used if this function is used to access an object window (that is, a window that is descended from `HWND_OBJECT`). This is because it is very possible that the thread that owns the object window might not be processing it's message queue, and if this is the case, a system halt could occur.

WinQueryWindowText - Errors

Possible returns from [WinGetLastError](#)

`PMERR_INVALID_HWND (0x1001)`

An invalid window handle was specified.

WinQueryWindowText - Related Functions

Related Functions

- [WinQueryDlgItemShort](#)
- [WinQueryDlgItemText](#)
- [WinQueryDlgItemTextLength](#)
- [WinQueryWindowText](#)
- [WinQueryWindowTextLength](#)
- [WinSetDlgItemShort](#)
- [WinSetDlgItemText](#)
- [WinSetWindowText](#)

WinQueryWindowText - Related Messages

Related Messages

- [WM_QUERYWINDOWPARAMS](#)

WinQueryWindowText - Example Code

This example shows how to query window text.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
#define FID_CLIENT 255

HWND hwndFrame;
HWND hwndClient;
char szTitle[32];

/* This function creates a new window of      */
/* class Generic and returns hwnd.           */

hwndClient = WinCreateWindow(hwndFrame,
                             "Generic",
                             (PSZ)"My Window", /* No window text      */
                             0UL,               /* No window style          */
                             0,0,0,0,           /* Position and size        */
                             (HWND)NULL,        /* No owner                 */
                             HWND_TOP,          /* On top of siblings       */
                             FID_CLIENT,        /* Client window ID         */
                             NULL,             /* Control data             */
                             NULL);            /* Pres. params             */

WinQueryWindowText(hwndFrame, sizeof(szTitle), szTitle);
```

WinQueryWindowText - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinQueryWindowTextLength

WinQueryWindowTextLength - Syntax

This call returns the length of the window text, excluding any null termination character.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd;      /* Window handle. */
LONG    lRetLen;    /* Length of the window text. */

lRetLen = WinQueryWindowTextLength(hwnd);
```

WinQueryWindowTextLength Parameter - hwnd

hwnd ([HWND](#)) - input
Window handle.

WinQueryWindowTextLength Return Value - lRetLen

lRetLen ([LONG](#)) - returns
Length of the window text.

WinQueryWindowTextLength - Parameters

hwnd ([HWND](#)) - input
Window handle.

lRetLen ([LONG](#)) - returns
Length of the window text.

WinQueryWindowTextLength - Remarks

This function sends a [WM_QUERYWINDOWPARAMS](#) message to *hwnd*.

WinQueryWindowTextLength - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinQueryWindowTextLength - Related Functions

Related Functions

- [WinQueryDlgItemShort](#)
- [WinQueryDlgItemText](#)
- [WinQueryDlgItemTextLength](#)
- [WinQueryWindowText](#)
- [WinQueryWindowTextLength](#)
- [WinSetDlgItemShort](#)
- [WinSetDlgItemText](#)
- [WinSetWindowText](#)

WinQueryWindowTextLength - Related Messages

Related Messages

- [WM_QUERYWINDOWPARAMS](#)

WinQueryWindowTextLength - Example Code

This example shows how to get the title-bar window text.

```
#define INCL_WINWINDOWMGR
#define INCL_DOSMEMMGR
#include <OS2.H>

HWND    hwndFrame;
PSZ     szTitle;
ULONG   cbBytes;

cbBytes = WinQueryWindowTextLength(hwndFrame);

DosAllocMem((PPVOID)szTitle,
            (ULONG)cbBytes,
            PAG_READ |
            PAG_WRITE |
            PAG_COMMIT);

WinQueryWindowText(hwndFrame, sizeof(szTitle), szTitle);
```

WinQueryWindowTextLength - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)

[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinQueryWindowThunkProc

WinQueryWindowThunkProc - Syntax

This function queries the pointer-conversion procedure associated with a window.

```
#define INCL_WINTHUNKAPI /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND     hwnd; /* Window handle. */
PFN      thunkpr; /* Pointer-conversion procedure identifier. */

thunkpr = WinQueryWindowThunkProc(hwnd);
```

WinQueryWindowThunkProc Parameter - hwnd

hwnd (**HWND**) - input
Window handle.

WinQueryWindowThunkProc Return Value - thunkpr

thunkpr (**PFN**) - returns
Pointer-conversion procedure identifier.

NULL	No pointer-conversion procedure is associated with this window.
Other	Identifier of the pointer-conversion procedure associated with this window.

WinQueryWindowThunkProc - Parameters

hwnd ([HWND](#)) - input
Window handle.

thunkpr ([PFN](#)) - returns
Pointer-conversion procedure identifier.

NULL	No pointer-conversion procedure is associated with this window.
Other	Identifier of the pointer-conversion procedure associated with this window.

WinQueryWindowThunkProc - Related Functions

Related Functions

- [WinQueryClassThunkProc](#)
- [WinQueryWindowModel](#)
- [WinQueryWindowThunkProc](#)
- [WinSetClassThunkProc](#)
- [WinSetWindowThunkProc](#)

WinQueryWindowThunkProc - Example Code

This example shows how to get pointer conversion procedure associated with the frame window.

```
#define INCL_WINTHUNKAPI
#include <OS2.H>

HWND hwndFrame;
PFN pthnkproc;

pthnkproc = WinQueryWindowThunkProc(hwndFrame);
```

WinQueryWindowThunkProc - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Example Code](#)
- [Related Functions](#)
- [Glossary](#)

WinQueryWindowULong

WinQueryWindowULong - Syntax

This function obtains the unsigned long integer value, at a specified offset, from the memory of a reserved window word, of a given window.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd;        /* Handle of window to be queried. */
LONG     index;       /* Zero-based index into the window words of the value to be queried. */
ULONG    ulValue;     /* Value contained in the window word. */

ulValue = WinQueryWindowULong(hwnd, index);
```

WinQueryWindowULong Parameter - hwnd

hwnd ([HWND](#)) - input
Handle of window to be queried.

WinQueryWindowULong Parameter - index

index ([LONG](#)) - input
Zero-based index into the window words of the value to be queried.

The units of **index** are bytes. Valid values are zero through (*cbWindowData* -4), where *cbWindowData* is the parameter in [WinRegisterClass](#) that specifies the number of bytes available for application-defined storage. Any of the QWL_* values are valid.

Note: QWS_* values cannot be used.

QWL_DEFBUTTON
The default push button for a dialog.

The default push button is the one that sends its [WM_COMMAND](#) message when the enter key is pressed.

QWL_HMQ
Handle of message queue of window. Note that the leading 16 bits of this value are zero.

QWL_HWNDFOCUSSAVE
Window handle of the child windows of this window that last possessed the focus when this frame window was last deactivated.

QWL_PENDATA
Reserved for use by operating system extensions. It allows an operating system extension to store data on a per window basis.

QWL_STYLE
Window style.

QWL_USER
A ULONG value for applications to use is present at offset QWL_USER in windows of the following preregistered window classes:

WC_BUTTON

WC_COMBOBOX
 WC_CONTAINER
 WC_ENTRYFIELD
 WC_FRAME (includes dialog windows)
 WC_LISTBOX
 WC_MENU
 WC_MLE
 WC_NOTEBOOK
 WC_SCROLLBAR
 WC_SLIDER
 WC_SPINBUTTON
 WC_STATIC
 WC_TITTLBAR
 WC_VALUESET

This value can be used to place application-specific data in controls.

Other

Zero-based index.

WinQueryWindowULong Return Value - ulValue

ulValue ([ULONG](#)) - returns

Value contained in the window word.

WinQueryWindowULong - Parameters

hwnd ([HWND](#)) - input

Handle of window to be queried.

index ([LONG](#)) - input

Zero-based index into the window words of the value to be queried.

The units of **index** are bytes. Valid values are zero through (*cbWindowData* -4), where *cbWindowData* is the parameter in [WinRegisterClass](#) that specifies the number of bytes available for application-defined storage. Any of the QWL_* values are valid.

Note: QWS_* values cannot be used.

QWL_DEFBUTTON

The default push button for a dialog.

The default push button is the one that sends its [WM_COMMAND](#) message when the enter key is pressed.

QWL_HMQ

Handle of message queue of window. Note that the leading 16 bits of this value are zero.

QWL_HWNDFOCUSSAVE

Window handle of the child windows of this window that last possessed the focus when this frame window was last deactivated.

QWL_PENDATA

Reserved for use by operating system extensions. It allows an operating system extension to store data on a per window basis.

QWL_STYLE

Window style.

QWL_USER

A ULONG value for applications to use is present at offset QWL_USER in windows of the following preregistered window classes:

WC_BUTTON
WC_COMBOBOX
WC_CONTAINER
WC_ENTRYFIELD
WC_FRAME (includes dialog windows)
WC_LISTBOX
WC_MENU
WC_MLE
WC_NOTEBOOK
WC_SCROLLBAR
WC_SLIDER
WC_SPINBUTTON
WC_STATIC
WC_TITTLBAR
WC_VALUESET

This value can be used to place application-specific data in controls.

Other

Zero-based index.

ulValue ([ULONG](#)) - returns
Value contained in the window word.

WinQueryWindowULong - Remarks

The window handle that is passed to this function can be the handle of a window with the same, or different, message queue as the caller, thereby allowing the caller to obtain data from windows belonging to other threads.

The specified *index* is valid only if all of the bytes referenced are within the reserved memory.

WinQueryWindowULong - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_PARAMETER_OUT_OF_RANGE (0x1003)
The value of a parameter was not within the defined valid range for that parameter.

WinQueryWindowULong - Related Functions

Related Functions

- [WinQueryWindowPtr](#)
- [WinQueryWindowULong](#)
- [WinQueryWindowUShort](#)
- [WinSetWindowBits](#)
- [WinSetWindowPtr](#)
- [WinSetWindowULong](#)
- [WinSetWindowUShort](#)

WinQueryWindowULong - Example Code

This example shows how to get the handle of the message queue of a window.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HWND hwnd;
HMQ hmq;

hmq = (HMQ)WinQueryWindowULong(hwnd, QWL_HMQ);
```

WinQueryWindowULong - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Errors](#)
- [Remarks](#)
- [Example Code](#)
- [Related Functions](#)
- [Glossary](#)

WinQueryWindowUShort

WinQueryWindowUShort - Syntax

This function obtains the unsigned short integer value at a specified offset from the reserved window word's memory of a given window.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd;      /* Handle of window to be queried. */
LONG    index;     /* Zero-based index into the window words of the value to be queried. */
USHORT  usValue;    /* Value contained in the indicated window word. */

usValue = WinQueryWindowUShort(hwnd, index);
```

WinQueryWindowUShort Parameter - hwnd

hwnd (**HWND**) - input
Handle of window to be queried.

WinQueryWindowUShort Parameter - index

index (**LONG**) - input
Zero-based index into the window words of the value to be queried.

The units of *index* are bytes. Valid values are zero through (*cbWindowData* -2), where *cbWindowData* is the parameter in [WinRegisterClass](#) that specifies the number of bytes available for application-defined storage. Any of the QWS_* values are valid.

Note: QWL_* values cannot be used.

QWS_CXRESTORE
The width to which the window is restored.

See also the QWS_CYRESTORE value.

QWS_CYRESTORE
The height to which the window is restored.

These values are only valid while the window is maximized or minimized (that is, while either the WS_MINIMIZED or WS_MAXIMIZED window style indicators are set). Changing these values with the [WinSetWindowUShort](#) call alters the restore size and position.

QWS_FLAGS
These indicators apply only to frame or dialog windows, and contain combinations of the following indicators:

FF_ACTIVE	Frame window is displayed in the active state.
FF_DIALOGBOX	Frame window is being used as a dialog box.
FF_DLGDISMISSED	Dialog has been dismissed by the WinDismissDlg function.
FF_FLASHHILITE	Window is currently flashed. This indicator toggles with each flash.
FF_FLASHWINDOW	Frame window is flashing.
FF_OWNERDISABLED	Window's owner is disabled. This indicator is only set if the window and its owner are siblings.
FF_OWNERHIDDEN	Frame window is hidden as a result of its owner being hidden or minimized. This indicator is set only if the window and its owner are siblings.
FF_SELECTED	Frame window is selected.

QWS_ID
Window identity. The value of the *id* parameter of the [WinCreateWindow](#) function.

QWS_RESULT
Dialog-result parameter, as established by the [WinDismissDlg](#) function.

QWS_XMINIMIZE
The x-coordinate of the position to which the window is minimized. If this value is -1, the window has not been minimized.

See also the QWS_YMINIMIZE value.

QWS_XRESTORE

The x-coordinate of the position to which the window is restored.

See also the QWS_CYRESTORE value.

QWS_YMINIMIZE

The y-coordinate of the position to which the window is minimized.

When the window is minimized for the first time an arbitrary position is chosen. Changing these values with the [WinSetWindowUShort](#) call alters the position of the minimized window, but only when the window is not in a minimized state.

QWS_YRESTORE

The y-coordinate of the position to which the window is restored.

See also the QWS_CYRESTORE value.

Other

Zero-based index.

WinQueryWindowUShort Return Value - usValue

usValue ([USHORT](#)) - returns

Value contained in the indicated window word.

WinQueryWindowUShort - Parameters

hwnd ([HWND](#)) - input

Handle of window to be queried.

index ([LONG](#)) - input

Zero-based index into the window words of the value to be queried.

The units of *index* are bytes. Valid values are zero through (*cbWindowData* -2), where *cbWindowData* is the parameter in [WinRegisterClass](#) that specifies the number of bytes available for application-defined storage. Any of the QWS_* values are valid.

Note: QWL_* values cannot be used.

QWS_CXRESTORE

The width to which the window is restored.

See also the QWS_CYRESTORE value.

QWS_CYRESTORE

The height to which the window is restored.

These values are only valid while the window is maximized or minimized (that is, while either the WS_MINIMIZED or WS_MAXIMIZED window style indicators are set). Changing these values with the [WinSetWindowUShort](#) call alters the restore size and position.

QWS_FLAGS

These indicators apply only to frame or dialog windows, and contain combinations of the following indicators:

FF_ACTIVE

Frame window is displayed in the active state.

FF_DIALOGBOX

Frame window is being used as a dialog box.

FF_DLGDISMISSED

Dialog has been dismissed by the [WinDismissDlg](#) function.

FF_FLASHHILITE	Window is currently flashed. This indicator toggles with each flash.
FF_FLASHWINDOW	Frame window is flashing.
FF_OWNERDISABLED	Window's owner is disabled. This indicator is only set if the window and its owner are siblings.
FF_OWNERHIDDEN	Frame window is hidden as a result of its owner being hidden or minimized. This indicator is set only if the window and its owner are siblings.
FF_SELECTED	Frame window is selected.
QWS_ID	Window identity. The value of the <i>id</i> parameter of the WinCreateWindow function.
QWS_RESULT	Dialog-result parameter, as established by the WinDismissDlg function.
QWS_XMINIMIZE	The x-coordinate of the position to which the window is minimized. If this value is -1, the window has not been minimized. See also the QWS_YMINIMIZE value.
QWS_XRESTORE	The x-coordinate of the position to which the window is restored. See also the QWS_CYRESTORE value.
QWS_YMINIMIZE	The y-coordinate of the position to which the window is minimized. When the window is minimized for the first time an arbitrary position is chosen. Changing these values with the WinSetWindowUShort call alters the position of the minimized window, but only when the window is not in a minimized state.
QWS_YRESTORE	The y-coordinate of the position to which the window is restored. See also the QWS_CYRESTORE value.
Other	Zero-based index.
usValue (USHORT) - returns Value contained in the indicated window word.	

WinQueryWindowUShort - Remarks

The window handle that is passed to this function can be the handle of a window with the same, or different, message queue as the caller, thereby allowing the caller to obtain data from windows belonging to other threads.

WinQueryWindowUShort - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_PARAMETER_OUT_OF_RANGE (0x1003)

The value of a parameter was not within the defined valid range for that parameter.

WinQueryWindowUShort - Related Functions

Related Functions

- [WinQueryWindowPtr](#)
- [WinQueryWindowULong](#)
- [WinQueryWindowUShort](#)
- [WinSetWindowBits](#)
- [WinSetWindowPtr](#)
- [WinSetWindowULong](#)
- [WinSetWindowUShort](#)

WinQueryWindowUShort - Related Messages

Related Messages

- [WM_COMMAND](#)

WinQueryWindowUShort - Example Code

In this example, the WinQueryWindowUShort call is used to query the window words to see if a window has been minimized.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HWND    hwnd;
USHORT  usResult;

usResult = WinQueryWindowUShort(hwnd, QWS_XMINIMIZE);
if (-1 == (LONG)usResult)
{
    /* Window has not been minimized */
}
```

WinQueryWindowUShort - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)

[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinRealizePalette

WinRealizePalette - Syntax

This function indicates that drawing is about to take place after a palette has been selected.

```
#define INCL_WIN /* Or use INCL_PM, */  
#include <os2.h>  
  
HWND      hwnd;      /* Window handle where drawing is taking place. */  
HPS       hps;       /* Presentation-space handle. */  
PULONG    pcclr;     /* Number of physical palette entries changed. */  
LONG      lChanged;  /* Number of colors remapped. */  
  
lChanged = WinRealizePalette(hwnd, hps, pcclr);
```

WinRealizePalette Parameter - hwnd

hwnd ([HWND](#)) - input
Window handle where drawing is taking place.

WinRealizePalette Parameter - hps

hps ([HPS](#)) - input
Presentation-space handle.

WinRealizePalette Parameter - pcclr

pcclr ([PULONG](#)) - output
Number of physical palette entries changed.

A value of zero indicates that the palette was successfully realized without changing any entries in the display hardware physical

table. A non-zero value gives the number of hardware table entries that were changed and indicates that a [WM_REALIZEPALETTE](#) message has been posted to all other applications.

WinRealizePalette Return Value - IChanged

IChecked (LONG) - returns

Number of colors remapped.

PAL_ERROR

Error occurred

Otherwise

Number of colors that are remapped. This includes both animating and non-animating indexes that have matches in the physical palette. This information can be used to determine whether the window needs repainting.

Note that this information may already be out of date if there are other palette-using applications running.

WinRealizePalette - Parameters

hwnd (HWND) - input

Window handle where drawing is taking place.

hps (HPS) - input

Presentation-space handle.

pcclr (PULONG) - output

Number of physical palette entries changed.

A value of zero indicates that the palette was successfully realized without changing any entries in the display hardware physical table. A non-zero value gives the number of hardware table entries that were changed and indicates that a [WM_REALIZEPALETTE](#) message has been posted to all other applications.

IChecked (LONG) - returns

Number of colors remapped.

PAL_ERROR

Error occurred

Otherwise

Number of colors that are remapped. This includes both animating and non-animating indexes that have matches in the physical palette. This information can be used to determine whether the window needs repainting.

Note that this information may already be out of date if there are other palette-using applications running.

WinRealizePalette - Remarks

This function is typically used after GpiSelectPalette or in response to a [WM_REALIZEPALETTE](#) message. It causes the system to ensure that the palette is appropriately realized for all drawing operations.

When the window has the input focus, the palette will be realized absolutely. Otherwise, the realization is on a best-can-do basis. If the palette is larger than the currently associated device can support, as many entries as possible are realized, starting from the lowest index.

If the presentation space is currently associated with a device context of type OD_MEMORY (see [DevOpenDC](#)), then this function performs no function other than returning without error.

This function must not be called while processing a [WM_SETFOCUS](#) message, because a window's activation state is not known until processing of this message is complete.

Note that the palette cannot be physically changed on all devices. The effect of this call is therefore device dependent.

WinRealizePalette - Errors

Possible returns from [WinGetLastError](#)

PMERR_NO_PALETTE_SELECTED (0x2110)

An attempt to realize a palette failed because no palette was previously selected into the Presentation Space.

PMERR_INVALID_HWND (0x1001)

An invalid window handle was specified.

PMERR_INV_HDC (0x207C)

An invalid device-context handle or (micro presentation space) presentation-space handle was specified.

PMERR_HDC_BUSY (0x2033)

An internal device context busy error was detected. The device context was locked by one thread during an attempt to access it from another thread.

PMERR_INV_IN_AREA (0x2085)

An attempt was made to issue a function invalid inside an area bracket. This can be detected while the actual drawing mode is **draw** or **draw-and-retain** or during segment drawing or correlation functions.

WinRealizePalette - Related Functions

Related Functions

- [WinBeginPaint](#)
- [WinEnableWindowUpdate](#)
- [WinEndPaint](#)
- [WinExcludeUpdateRegion](#)
- [WinGetClipPS](#)
- [WinGetPS](#)
- [WinGetScreenPS](#)
- [WinInvalidateRect](#)
- [WinInvalidateRegion](#)
- [WinIsWindowShowing](#)
- [WinIsWindowVisible](#)
- [WinLockVisRegions](#)
- [WinOpenWindowDC](#)
- [WinQueryUpdateRect](#)
- [WinQueryUpdateRegion](#)
- [WinRealizePalette](#)
- [WinReleasePS](#)
- [WinShowWindow](#)
- [WinUpdateWindow](#)
- [WinValidateRect](#)
- [WinValidateRegion](#)

WinRealizePalette - Related Messages

Related Messages

- [WM_SETFOCUS](#)
- [WM_REALIZEPALETTE](#)

WinRealizePalette - Example Code

In this example, the WinRealizePalette call is issued in response to a WM_REALIZEPALETTE. This ensures that the palette is appropriately realized for all drawing operations.

```
#define INCL_WIN
#include <OS2.H>
HWND hwnd;
ULONG cclr;
USHORT msg;
HPS hps;

switch(msg)
{
    case WM_REALIZEPALETTE:

        WinRealizePalette(hwnd,hps,&cclr);
}
```

WinRealizePalette - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinRegisterClass

WinRegisterClass - Syntax

This function registers a window class.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
```

```
#include <os2.h>

HAB      hab;           /* Anchor-block handle. */
PSZ      pszClassName;  /* Window-class name. */
PFNWP    pfnWndProc;    /* Window-procedure identifier. */
ULONG    flStyle;       /* Default-window style. */
ULONG    cbWindowData;  /* Reserved storage. */
BOOL     rc;            /* Window-class-registration indicator. */

rc = WinRegisterClass(hab, pszClassName, pfnWndProc,
                     flStyle, cbWindowData);
```

WinRegisterClass Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinRegisterClass Parameter - pszClassName

pszClassName (**PSZ**) - input
Window-class name.

An application-specified class name.

WinRegisterClass Parameter - pfnWndProc

pfnWndProc (**PFNWP**) - input
Window-procedure identifier.

Can be NULL if the application does not provide its own window procedure.

WinRegisterClass Parameter - flStyle

flStyle (**ULONG**) - input
Default-window style.

This can be any of the standard class styles (CS_*) in addition to any class-specific styles that are defined. These styles can be augmented when a window of this class is created.

A public window class is created if the CS_PUBLIC style is specified, otherwise a private class is created. The CS_PUBLIC style must only be specified for the shell process.

Public classes are available for creating windows from any process. Private classes are only available to the registering process.

WinRegisterClass Parameter - cbWindowData

cbWindowData ([ULONG](#)) - input
Reserved storage.

This is the number of bytes of storage reserved per window created of this class for application use.

WinRegisterClass Return Value - rc

rc ([BOOL](#)) - returns
Window-class-registration indicator.

TRUE	Window class successfully registered
FALSE	Window class not successfully registered.

WinRegisterClass - Parameters

hAb ([HAB](#)) - input
Anchor-block handle.

pszClassName ([PSZ](#)) - input
Window-class name.

An application-specified class name.

pfnWndProc ([PFNWP](#)) - input
Window-procedure identifier.

Can be NULL if the application does not provide its own window procedure.

flStyle ([ULONG](#)) - input
Default-window style.

This can be any of the standard class styles (CS_*) in addition to any class-specific styles that are defined. These styles can be augmented when a window of this class is created.

A public window class is created if the CS_PUBLIC style is specified, otherwise a private class is created. The CS_PUBLIC style must only be specified for the shell process.

Public classes are available for creating windows from any process. Private classes are only available to the registering process.

cbWindowData ([ULONG](#)) - input
Reserved storage.

This is the number of bytes of storage reserved per window created of this class for application use.

rc ([BOOL](#)) - returns
Window-class-registration indicator.

TRUE

FALSE	Window class successfully registered
	Window class not successfully registered.

WinRegisterClass - Remarks

When an application registers a private class with the window procedure in a dynamic link library, it is the application's responsibility to resolve the window-procedure address before issuing this function.

A private class must not be registered with the same name as a public class in the same process.

However, if a private class is registered with the same name as one that already exists, the parameters replace the old class parameters, and the return value is TRUE. The window procedure of an existing window can be changed using [WinSubclassWindow](#) or [WinSetWindowPtr](#). The style of an existing window can be changed with the [WinSetWindowULong](#) or [WinSetWindowUShort](#) functions. The number of bytes of storage allocated for application use cannot be changed once the window is created.

Private classes are deleted when the process that registers them terminates.

WinRegisterClass - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_FLAG (0x1019)

An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

PMERR_INVALID_INTEGER_ATOM (0x1016)

The specified atom is not a valid integer atom.

PMERR_INVALID_HATOMTBL (0x1013)

An invalid atom-table handle was specified.

PMERR_INVALID_ATOM_NAME (0x1015)

An invalid atom name string was passed.

PMERR_ATOM_NAME_NOT_FOUND (0x1017)

The specified atom name is not in the atom table.

WinRegisterClass - Related Functions

Related Functions

- [WinCalcFrameRect](#)
- [WinCreateFrameControls](#)
- [WinCreateStdWindow](#)
- [WinCreateWindow](#)
- [WinDefWindowProc](#)
- [WinDestroyWindow](#)
- [WinQueryClassInfo](#)
- [WinQueryClassName](#)
- [WinRegisterClass](#)
- [WinSubclassWindow](#)

WinRegisterClass - Example Code

This example calls WinRegisterClass to register a class or returns FALSE if an error occurs.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
HAB hab;
CHAR szClassName[] = "Generic"; /* window class name */
PFNWP pGenericWndProc;

if (!WinRegisterClass(hab, /* anchor-block handle */
    szClassName, /* class name */
    pGenericWndProc, /* window procedure */
    0L, /* window style */
    0)) /* amount of reserved memory */
    return (FALSE);
```

WinRegisterClass - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinRegisterObjectClass

WinRegisterObjectClass - Syntax

The WinRegisterObjectClass function registers a workplace object class.

```
#define INCL_WINWORKPLACE
#include <os2.h>

PSZ pClassName; /* A pointer to object class being registered. */
PSZ pModname; /* A pointer to DLL name. */
BOOL rc; /* Success indicator. */

rc = WinRegisterObjectClass(pClassName, pModname);
```

WinRegisterObjectClass Parameter - pClassName

pClassName (PSZ) - input
A pointer to object class being registered.

A pointer to a zero-terminated string which contains the name of the object class being registered in the workplace.

WinRegisterObjectClass Parameter - pModname

pModname (PSZ) - input
A pointer to DLL name.

A pointer to a zero-terminated string which contains the name of the DLL which holds the object definition.

WinRegisterObjectClass Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinRegisterObjectClass - Parameters

pClassName (PSZ) - input
A pointer to object class being registered.

A pointer to a zero-terminated string which contains the name of the object class being registered in the workplace.

pModname (PSZ) - input
A pointer to DLL name.

A pointer to a zero-terminated string which contains the name of the DLL which holds the object definition.

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinRegisterObjectClass - Remarks

The DLL must be one created using the IBM System Object Model. Object classes will automatically be added to the system when installing a DLL which contains an object definition. Generally, it is not required for the object DLL to be present at the time WinRegisterObjectClass is called. However, if the object class overrides wpclsQueryInstanceType or wpclsQueryInstanceFilter, the DLL must be present at the time of the class registration.

WinRegisterObjectClass - Related Functions

Related Functions

- [WinCreateObject](#)
 - [WinDeregisterObjectClass](#)
 - [WinReplaceObjectClass](#)
-

WinRegisterObjectClass - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

WinRegisterUserDatatype

WinRegisterUserDatatype - Syntax

This function registers a data type and defines its structure.

```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
LONG      datatype;     /* Data type code to be defined. */
LONG      count;        /* Number of elements. */
PLONG     types;        /* Data type codes of structure components. */
BOOL      rc;           /* Success indicator. */

rc = WinRegisterUserDatatype(hab, datatype,
                             count, types);
```

WinRegisterUserDatatype Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinRegisterUserDatatype Parameter - datatype

datatype ([LONG](#)) - input
Data type code to be defined.

This must not be less than DTYP_USER, and must not have been defined previously.

WinRegisterUserDatatype Parameter - count

count ([LONG](#)) - input
Number of elements.

Must not be less than one.

WinRegisterUserDatatype Parameter - types

types ([PLONG](#)) - input
Data type codes of structure components.

Valid data types are the system-defined data types and their pointer equivalents, application-defined data types and their pointer equivalents, and control data types. Note that not all of the data types that occur in the CPI can be specified on this function.

A control data type is followed by one or more entries in the *types* array that are interpreted in a special way. Control data types allow arrays, offsets, and lengths to be defined.

Simple Data Types:

DTYP_ATOM	See ATOM .
DTYP_BIT16	See USHORT .
DTYP_BIT32	See ULONG .
DTYP_BIT8	See UCHAR .
DTYP_BOOL	See BOOL .
DTYP_COUNT2	See USHORT .
DTYP_COUNT2B	See USHORT .
DTYP_COUNT2CH	See USHORT .
DTYP_COUNT4B	See ULONG .
DTYP_CPID	See USHORT .
DTYP_ERRORID	See ERRORID .
DTYP_IDENTITY	See USHORT .
DTYP_IDENTITY4	See ULONG .
DTYP_INDEX2	See USHORT .
DTYP_IPT	See IPT .

DTYP_LENGTH2
DTYP_LENGTH4
DTYP_LONG
DTYP_OFFSET2B
DTYP_PID
DTYP_PIX
DTYP_PROGCATEGORY
DTYP_PROPERTY2
DTYP_PROPERTY4
DTYP_RESID
DTYP_SEGOFF
DTYP_SHORT
DTYP_TID
DTYP_TIME
DTYP_UCHAR
DTYP_ULONG
DTYP_USHORT
DTYP_WIDTH4
DTYP_WNDPROC

Handle Data Types:

DTYP_HAB
DTYP_HACCEL
DTYP_HAPP
DTYP_HATOMTBL
DTYP_HBITMAP
DTYP_HDC
DTYP_HENUM
DTYP_HINI
DTYP_HLIB
DTYP_HMF
DTYP_HMQ
DTYP_HPOINTER
DTYP_HPROGRAM
DTYP_HPS
DTYP_HRGN
DTYP_HSEM
DTYP_HSPL
DTYP_HSWITCH
DTYP_HWND

DTYP_BYTE
DTYP_CHAR
DTYP_STRL
DTYP_STR16
DTYP_STR32
DTYP_STR64
DTYP_STR8

Structure Data Types:

DTYP_ACCEL
DTYP_ACCELTABLE
DTYP_ARCPARAMS
DTYP_AREABUNDLE
DTYP_BITMAPINFO
DTYP_BITMAPINFOHEADER
DTYP_BTNCDATA
DTYP_CATCHBUF
DTYP_CHARBUNDLE
DTYP_CLASSINFO
DTYP_CREATESTRUCT
DTYP_CURSORINFO
DTYP_DEVOPENSTRUC
DTYP_DLGTEMPLATE
DTYP_DLGITEM
DTYP_ENTRYFDATA
DTYP_FATTRS
DTYP_FFDESCS
DTYP_FIXED
DTYP_FONTMETRICS
DTYP_FRAMECDATA

See [USHORT](#).
See [ULONG](#).
See [LONG](#).
See [USHORT](#).
See [PID](#).
See [PIX](#).
See [PROGCATEGORY](#).
See [USHORT](#).
See [LONG](#).
See [HMODULE](#).
See [SEGOFF](#).
See [SHORT](#).
See [TID](#).
See [LONG](#).
See [UCHAR](#).
See [ULONG](#).
See [USHORT](#).
See [LONG](#).
See [PFNWP](#).

See [HAB](#).
See [HACCEL](#).
See [HAPP](#).
See [HATOMTBL](#).
See [HBITMAP](#).
See [HDC](#).
See [HENUM](#).
See [HINI](#).
See [HLIB](#).
See [HMF](#).
See [HMQ](#).
See [HPOINTER](#).
See [HPROGRAM](#).
See [HPS](#).
See [HRGN](#).
See [HSEM](#).
See [HSPL](#).
See [HSWITCH](#).
See [HWND](#).

Character/String/Buffer Data Types:

See [BYTE](#).
See [CHAR](#).
See [PSZ](#).
See [STR16](#).
See [STR32](#).
See [STR64](#).
See [STR8](#).

See [ACCEL](#).
See [ACCELTABLE](#).
See [ARCPARAMS](#).
See [AREABUNDLE](#).
See [BITMAPINFO](#).
See [BITMAPINFOHEADER](#).
See [BTNCDATA](#).
See [CATCHBUF](#).
See [CHARBUNDLE](#).
See [CLASSINFO](#).
See [CREATESTRUCT](#).
See [CURSORINFO](#).
See [DEVOPENSTRUC](#).
See [DLGTEMPLATE](#).
See [DLGITEM](#).
See [ENTRYFDATA](#).
See [FATTRS](#).
See [FFDESCS](#).
See [FIXED](#).
See [FONTMETRICS](#).
See [FRAMECDATA](#).

DTYP_GRADIENTL
DTYP_HCINFO
DTYP_IMAGEBUNDLE
DTYP_KERNINGPAIRS
DTYP_LINEBUNDLE
DTYP_MARGSTRUCT
DTYP_MARKERBUNDLE
DTYP_MATRIXLF
DTYP_MLECTLDATA
DTYP_OVERFLOW
DTYP_OWNERITEM
DTYP_POINTERINFO
DTYP_POINTL
DTYP_PROGRAMENTRY
DTYP_PROGTYPE
DTYP_QMSG
DTYP_RECTL
DTYP_RGB
DTYP_RGNRECT
DTYP_SBCDATA
DTYP_SIZEF
DTYP_SIZEL
DTYP_SWBLOCK
DTYP_SWCNTRL
DTYP_SWENTRY
DTYP_SWP
DTYP_TRACKINFO
DTYP_USERBUTTON
DTYP_WNDPARAMS
DTYP_WPOINT
DTYP_WRECT
DTYP_XYWINSIZE

Pointer Data Type:

DTYP_P*

See [GRADIENTL](#).
See [HCINFO](#).
See [IMAGEBUNDLE](#).
See [KERNINGPAIRS](#).
See [LINEBUNDLE](#).
See [MLEMARGSTRUCT](#).
See [MARKERBUNDLE](#).
See [MATRIXLF](#).
See [MLECTLDATA](#).
See [MLEOVERFLOW](#).
See [OWNERITEM](#).
See [POINTERINFO](#).
See [POINTL](#).
See [PROGRAMENTRY](#).
See [PROGTYPE](#).
See [QMSG](#).
See [RECTL](#).
See [RGB](#).
See [RGNRECT](#).
See [SBCDATA](#).
See [SIZEF](#).
See [SIZEL](#).
See [SWBLOCK](#).
See [SWCNTRL](#).
See [SWENTRY](#).
See [SWP](#).
See [TRACKINFO](#).
See [USERBUTTON](#).
See [WNDPARAMS](#).
See [WPOINT](#).
See [WRECT](#).
See [XYWINSIZE](#).

Pointer to an item of data type DTYP_*, where DTYP_* is one of the data types in the preceding lists. The value of a pointer data type is the value of the corresponding non-pointer data type prefixed with a minus to make it negative.

Minimum Application Data Type:

DTYP_USER

Minimum value for application-defined non-pointer data types.

Control Data Type:

DTYP_CTL_ARRAY

This starts a sequence of three array elements that define an array; the array resides in the structure being defined, and may have a fixed number of elements or a variable number of elements.

The following describes the possible elements:

n	DTYP_CTL_ARRAY
n+1	data type of array data.
n+2	minus the number of elements in the array (for an array of fixed size), or the index of the element in <i>types</i> corresponding to the structure component which contains the number of elements in the array being defined. This component must have a suitable numeric data type. The array-size element must precede element "n" in <i>types</i> . The index is zero-based.

DTYP_CTL_LENGTH

This starts a sequence of four array elements that define a structure component containing the length of part or all of the structure. The length component resides at this point in the structure.

The following describes the possible elements:

n	DYP_CTL_LENGTH
---	----------------

	n+1	data type of structure component that contains the length (must be a suitable numeric data type).														
	n+2	<p>the index of the element in <i>types</i> corresponding to the first structure component that is included in the length; a value of -1 denotes the start of the structure. This index is zero-based.</p> <p>The element specified must not be one that is the second or subsequent element in a DTYP_CTL_* sequence of elements.</p>														
	N+3	<p>the index of the element in <i>types</i> corresponding to the last structure component that is included in the length; it must not be less than the value contained in element n+2. A value of -1 denotes the end of the structure. The index is zero-based.</p> <p>The element specified must not be one that is the second or subsequent element in a DTYP_CTL_* sequence of elements.</p> <p>If the value is -1, the length includes all offset data residing at the end of the structure.</p>														
DTYP_CTL_OFFSET	<p>This starts a sequence of four array elements that define data addressed by an offset. The offset resides at this point in the structure, and contains the offset in bytes of the data from the start of the <i>outermost</i> structure in which this component resides. The data addressed by the offset must occupy storage following the fixed part of the structure. The data may be scalar data or array data.</p> <p>The following describes the possible elements:</p> <table><tr><td>n</td><td>DTYP_CTL_OFFSET</td></tr><tr><td>n+1</td><td>data type of the structure component that contains the offset (must be a suitable unsigned numeric data type).</td></tr><tr><td>n+2</td><td>data type of offset data.</td></tr><tr><td>n+3</td><td>minus the number of elements in the array (for an array of fixed size), or the index of the element in <i>types</i> corresponding to the structure component that contains the number of elements in the array being defined; this component must have a suitable numeric data type; the array-size element may occur earlier or later in the structure. This index is zero-based.</td></tr></table> <p>A value of -1 indicates that the data is not an array.</p> <p>This starts a sequence of three array elements that define a pointer to an array. The pointer resides at this point in the structure, and the array resides elsewhere. The array can have a fixed or variable number of elements.</p> <p>The following describes the possible elements:</p> <table><tr><td>n</td><td>DTYP_CTL_PARRAY</td></tr><tr><td>n+1</td><td>data type of array data.</td></tr><tr><td>n+2</td><td>minus the number of elements in the</td></tr></table>		n	DTYP_CTL_OFFSET	n+1	data type of the structure component that contains the offset (must be a suitable unsigned numeric data type).	n+2	data type of offset data.	n+3	minus the number of elements in the array (for an array of fixed size), or the index of the element in <i>types</i> corresponding to the structure component that contains the number of elements in the array being defined; this component must have a suitable numeric data type; the array-size element may occur earlier or later in the structure. This index is zero-based.	n	DTYP_CTL_PARRAY	n+1	data type of array data.	n+2	minus the number of elements in the
n	DTYP_CTL_OFFSET															
n+1	data type of the structure component that contains the offset (must be a suitable unsigned numeric data type).															
n+2	data type of offset data.															
n+3	minus the number of elements in the array (for an array of fixed size), or the index of the element in <i>types</i> corresponding to the structure component that contains the number of elements in the array being defined; this component must have a suitable numeric data type; the array-size element may occur earlier or later in the structure. This index is zero-based.															
n	DTYP_CTL_PARRAY															
n+1	data type of array data.															
n+2	minus the number of elements in the															
DTYP_CTL_PARRAY																

array (for an array of fixed size) or the index of the element in *types* corresponding to the structure component that contains the number of elements in the array being defined. This component must have a suitable numeric data type. The array-size element may occur earlier or later in the structure. The index is zero-based.

WinRegisterUserDatatype Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinRegisterUserDatatype - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

datatype ([LONG](#)) - input
Data type code to be defined.

This must not be less than DTYP_USER, and must not have been defined previously.

count ([LONG](#)) - input
Number of elements.

Must not be less than one.

types ([PLONG](#)) - input
Data type codes of structure components.

Valid data types are the system-defined data types and their pointer equivalents, application-defined data types and their pointer equivalents, and control data types. Note that not all of the data types that occur in the CPI can be specified on this function.

A control data type is followed by one or more entries in the *types* array that are interpreted in a special way. Control data types allow arrays, offsets, and lengths to be defined.

Simple Data Types:

DTYP_ATOM	See ATOM .
DTYP_BIT16	See USHORT .
DTYP_BIT32	See ULONG .
DTYP_BIT8	See UCHAR .
DTYP_BOOL	See BOOL .
DTYP_COUNT2	See USHORT .
DTYP_COUNT2B	See USHORT .
DTYP_COUNT2CH	See USHORT .
DTYP_COUNT4B	See ULONG .
DTYP_CPID	See USHORT .
DTYP_ERRORID	See ERRORID .
DTYP_IDENTITY	See USHORT .

DTYP_IDENTITY4
DTYP_INDEX2
DTYP_IPT
DTYP_LENGTH2
DTYP_LENGTH4
DTYP_LONG
DTYP_OFFSET2B
DTYP_PID
DTYP_PIX
DTYP_PROGCATEGORY
DTYP_PROPERTY2
DTYP_PROPERTY4
DTYP_RESID
DTYP_SEGOFF
DTYP_SHORT
DTYP_TID
DTYP_TIME
DTYP_UCHAR
DTYP_ULONG
DTYP_USHORT
DTYP_WIDTH4
DTYP_WNDPROC

Handle Data Types:

DTYP_HAB
DTYP_HACCEL
DTYP_HAPP
DTYP_HATOMTBL
DTYP_HBITMAP
DTYP_HDC
DTYP_HENUM
DTYP_HINI
DTYP_HLIB
DTYP_HMF
DTYP_HMQ
DTYP_HPOINTER
DTYP_HPROGRAM
DTYP_HPS
DTYP_HRGN
DTYP_HSEM
DTYP_HSPL
DTYP_HSWITCH
DTYP_HWND

DTYP_BYTE
DTYP_CHAR
DTYP_STRL
DTYP_STR16
DTYP_STR32
DTYP_STR64
DTYP_STR8

Structure Data Types:

DTYP_ACCEL
DTYP_ACCELTABLE
DTYP_ARCPARAMS
DTYP_AREABUNDLE
DTYP_BITMAPINFO
DTYP_BITMAPINFOHEADER
DTYP_BTNCDATA
DTYP_CATCHBUF
DTYP_CHARBUNDLE
DTYP_CLASSINFO
DTYP_CREATESTRUCT
DTYP_CURSORINFO
DTYP_DEVOPENSTRUC
DTYP_DLGTEMPLATE
DTYP_DLGITEM
DTYP_ENTRYFDATA
DTYP_FATTRS
DTYP_FFDESCS

See [ULONG](#).
See [USHORT](#).
See [IPT](#).
See [USHORT](#).
See [ULONG](#).
See [LONG](#).
See [USHORT](#).
See [PID](#).
See [PIX](#).
See [PROGCATEGORY](#).
See [USHORT](#).
See [LONG](#).
See [HMODULE](#).
See [SEGOFF](#).
See [SHORT](#).
See [TID](#).
See [LONG](#).
See [UCHAR](#).
See [ULONG](#).
See [USHORT](#).
See [LONG](#).
See [PFNWP](#).

See [HAB](#).
See [HACCEL](#).
See [HAPP](#).
See [HATOMTBL](#).
See [HBITMAP](#).
See [HDC](#).
See [HENUM](#).
See [HINI](#).
See [HLIB](#).
See [HMF](#).
See [HMQ](#).
See [HPOINTER](#).
See [HPROGRAM](#).
See [HPS](#).
See [HRGN](#).
See [HSEM](#).
See [HSPL](#).
See [HSWITCH](#).
See [HWN](#)D. **Character/String/Buffer Data Types:**

See [BYTE](#).
See [CHAR](#).
See [PSZ](#).
See [STR16](#).
See [STR32](#).
See [STR64](#).
See [STR8](#).

See [ACCEL](#).
See [ACCELTABLE](#).
See [ARCPARAMS](#).
See [AREABUNDLE](#).
See [BITMAPINFO](#).
See [BITMAPINFOHEADER](#).
See [BTNCDATA](#).
See [CATCHBUF](#).
See [CHARBUNDLE](#).
See [CLASSINFO](#).
See [CREATESTRUCT](#).
See [CURSORINFO](#).
See [DEVOPENSTRUC](#).
See [DLGTEMPLATE](#).
See [DLGITEM](#).
See [ENTRYFDATA](#).
See [FATTRS](#).
See [FFDESCS](#).

DTYP_FIXED
DTYP_FONTMETRICS
DTYP_FRAMECDATA
DTYP_GRADIENTL
DTYP_HCINFO
DTYP_IMAGEBUNDLE
DTYP_KERNINGPAIRS
DTYP_LINEBUNDLE
DTYP_MARGSTRUCT
DTYP_MARKERBUNDLE
DTYP_MATRIXLF
DTYP_MLECTLDATA
DTYP_OVERFLOW
DTYP_OWNERITEM
DTYP_POINTERINFO
DTYP_POINTL
DTYP_PROGRAMENTRY
DTYP_PROGTYPE
DTYP_QMSG
DTYP_RECTL
DTYP_RGB
DTYP_RGNRECT
DTYP_SBCDATA
DTYP_SIZEF
DTYP_SIZEL
DTYP_SWBLOCK
DTYP_SWCNTRL
DTYP_SWENTRY
DTYP_SWP
DTYP_TRACKINFO
DTYP_USERBUTTON
DTYP_WNDPARAMS
DTYP_WPOINT
DTYP_WRECT
DTYP_XYWINSIZE

Pointer Data Type:

DTYP_P*

See [FIXED](#).
See [FONTMETRICS](#).
See [FRAMECDATA](#).
See [GRADIENTL](#).
See [HCINFO](#).
See [IMAGEBUNDLE](#).
See [KERNINGPAIRS](#).
See [LINEBUNDLE](#).
See [MLEMARGSTRUCT](#).
See [MARKERBUNDLE](#).
See [MATRIXLF](#).
See [MLECTLDATA](#).
See [MLEOVERFLOW](#).
See [OWNERITEM](#).
See [POINTERINFO](#).
See [POINTL](#).
See [PROGRAMENTRY](#).
See [PROGTYPE](#).
See [QMSG](#).
See [RECTL](#).
See [RGB](#).
See [RGNRECT](#).
See [SBCDATA](#).
See [SIZEF](#).
See [SIZEL](#).
See [SWBLOCK](#).
See [SWCNTRL](#).
See [SWENTRY](#).
See [SWP](#).
See [TRACKINFO](#).
See [USERBUTTON](#).
See [WNDPARAMS](#).
See [WPOINT](#).
See [WRECT](#).
See [XYWINSIZE](#).

Pointer to an item of data type DTYP_*, where DTYP_* is one of the data types in the preceding lists. The value of a pointer data type is the value of the corresponding non-pointer data type prefixed with a minus to make it negative.

Minimum Application Data Type:

DTYP_USER

Minimum value for application-defined non-pointer data types.

Control Data Type:

DTYP_CTL_ARRAY

This starts a sequence of three array elements that define an array; the array resides in the structure being defined, and may have a fixed number of elements or a variable number of elements.

The following describes the possible elements:

n	DTYP_CTL_ARRAY
n +1	data type of array data.
n+2	minus the number of elements in the array (for an array of fixed size), or the index of the element in <i>types</i> corresponding to the structure component which contains the number of elements in the array being defined. This component must have a suitable numeric data type. The array-size element must precede element "n" in <i>types</i> . The index is zero-based.

DTYP_CTL_LENGTH

This starts a sequence of four array elements that define a structure component containing the length of part or all of the structure. The length component resides at this point in the structure.

The following describes the possible elements:	
n	DYP_CTL_LENGTH
n+1	data type of structure component that contains the length (must be a suitable numeric data type).
n+2	the index of the element in <i>types</i> corresponding to the first structure component that is included in the length; a value of -1 denotes the start of the structure. This index is zero-based. The element specified must not be one that is the second or subsequent element in a DTYP_CTL_* sequence of elements.
N+3	the index of the element in <i>types</i> corresponding to the last structure component that is included in the length; it must not be less than the value contained in element n+2. A value of -1 denotes the end of the structure. The index is zero-based. The element specified must not be one that is the second or subsequent element in a DTYP_CTL_* sequence of elements. If the value is -1, the length includes all offset data residing at the end of the structure.

DTYP_CTL_OFFSET

This starts a sequence of four array elements that define data addressed by an offset. The offset resides at this point in the structure, and contains the offset in bytes of the data from the start of the *outermost* structure in which this component resides. The data addressed by the offset must occupy storage following the fixed part of the structure. The data may be scalar data or array data.

The following describes the possible elements:	
n	DTYP_CTL_OFFSET
n+1	data type of the structure component that contains the offset (must be a suitable unsigned numeric data type).
n+2	data type of offset data.
n+3	minus the number of elements in the array (for an array of fixed size), or the index of the element in <i>types</i> corresponding to the structure component that contains the number of elements in the array being defined; this component must have a suitable numeric data type; the array-size element may occur earlier or later in the structure. This index is zero-based. A value of -1 indicates that the data is not an array.

DTYP_CTL_PARRAY

This starts a sequence of three array elements that define a pointer to an array. The pointer resides at this point in the structure, and the array resides elsewhere. The array can have a fixed or variable number of elements.

The following describes the possible elements:
n DTYP_CTL_PARRAY

n+1	data type of array data.
n+2	minus the number of elements in the array (for an array of fixed size) or the index of the element in <i>types</i> corresponding to the structure component that contains the number of elements in the array being defined. This component must have a suitable numeric data type. The array-size element may occur earlier or later in the structure. The index is zero-based.

rc (**BOOL**) - returns
 Success indicator.

TRUE Successful completion

FALSE Error occurred.

WinRegisterUserDatatype - Remarks

This function has no effect unless the [RegisterUserHook](#) hook has been set.

The value to be used should be obtained by calling [WinAddAtom](#) with the handle of the system atom manager, and subtracting DTYP_ATOM_OFFSET from the result.

[WinAddAtom](#) is guaranteed to return values in the range 0xC000 to 0xFFFF.

When a data type is defined using this function, a definition for the corresponding pointer data type is automatically established.

WinRegisterUserDatatype - Errors

Possible returns from [WinGetLastError](#)

PMERR_DATATYPE_TOO_SMALL (0x1048)
 The datatype specified was too small.

PMERR_DATATYPE_NOT_UNIQUE (0x1046)
 An attempt to register a datatype failed because it is not unique.

PMERR_ARRAY_TOO_SMALL (0x103E)
 The array specified was too small.

PMERR_DATATYPE_ENTRY_NOT_NUM (0x1043)
 The datatype entry specified was not numerical.

PMERR_DATATYPE_ENTRY_NOT_OFF (0x1044)
 The datatype entry specified was not an offset.

PMERR_DATATYPE_ENTRY_BAD_INDEX (0x103F)
 An invalid datatype entry index was specified.

PMERR_DATATYPE_ENTRY_CTL_MISS (0x1041)
 The datatype entry control was missing.

PMERR_DATATYPE_ENTRY_CTL_BAD (0x1040)
 An invalid datatype entry control was specified.

WinRegisterUserDatatype - Related Functions

Related Functions

- [WinBroadcastMsg](#)
- [WinCreateMsgQueue](#)
- [WinDestroyMsgQueue](#)
- [WinDispatchMsg](#)
- [WinGetDlgMsg](#)
- [WinGetMsg](#)
- [WinInSendMessage](#)
- [WinPeekMsg](#)
- [WinPostMsg](#)
- [WinPostQueueMsg](#)
- [WinQueryMsgPos](#)
- [WinQueryMsgTime](#)
- [WinQueryQueueInfo](#)
- [WinQueryQueueStatus](#)
- [WinSendDlgItemMsg](#)
- [WinSendMessage](#)
- [WinSetClassMsgInterest](#)
- [WinSetMsgInterest](#)
- [WinSetMsgMode](#)
- [WinSetSynchroMode](#)
- [WinWaitMsg](#)

WinRegisterUserDatatype - Example Code

This example calls WinRegisterUserDatatype to register a class or returns FALSE if an error occurs.

```
#define INCL_WINMESSAGEGR
#define INCL_WINTYPES
#include <OS2.H>
#define DTYP_MINE DTYP_USER + 1

HAB hab;
LONG asTypes[3] = {DTYP_CHAR,
                  DTYP_STRL,
                  DTYP_STR32};

WinRegisterUserDataTypes(hab,
                        DTYP_MINE,
                        3,
                        asTypes);
```

WinRegisterUserDatatype - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)

WinRegisterUserMsg

WinRegisterUserMsg - Syntax

This function registers a user message and defines its parameters.

```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
ULONG    msgid;         /* Message identifier. */
LONG     datatype1;     /* Data type of message parameter 1. */
LONG     dir1;          /* Direction of message parameter 1. */
LONG     datatype2;     /* Data type of message parameter 2. */
LONG     dir2;          /* Direction of message parameter 2. */
LONG     datatypeR;     /* Data type of message reply. */
BOOL     rc;            /* Success indicator. */

rc = WinRegisterUserMsg(hab, msgid, datatype1,
                        dir1, datatype2, dir2, datatypeR);
```

WinRegisterUserMsg Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinRegisterUserMsg Parameter - msgid

msgid (**ULONG**) - input
Message identifier.

This must not be less than WM_USER, and must not have been defined previously.

WinRegisterUserMsg Parameter - datatype1

datatype1 ([LONG](#)) - input
Data type of message parameter 1.

Valid data types are listed below. For data types that are shorter than 4 bytes, the data must be placed in the least-significant bytes, with the most significant bytes nullified (unsigned data types) or signed-extended (signed data types).

DTYP_BIT16	See BIT16 data type.
DTYP_BIT32	See BIT32 data type.
DTYP_BIT8	See BIT8 data type.
DTYP_BOOL	See BOOL data type.
DTYP_LONG	See LONG data type.
DTYP_SHORT	See SHORT data type.
DTYP_UCHAR	See UCHAR data type.
DTYP_ULONG	See ULONG data type.
DTYP_USHORT	See USHORT data type.
DTYP_P*	A pointer to a system data type. Note that not all of the system data types that exist in the CPI are valid.
< -DTYP_USER	A pointer to a user data type. The user data type must have already been defined via WinRegisterUserDatatype .

WinRegisterUserMsg Parameter - dir1

dir1 ([LONG](#)) - input
Direction of message parameter 1.

If the message parameter is a pointer, the direction values listed below apply to the *contents* of the storage location pointed at, as well as to the message parameter itself.

RUM_IN	Input parameter (inspected by the recipient of the message, but not altered)
RUM_OUT	Output parameter (altered by the recipient of the message, without inspecting its value first)
RUM_INOUT	Input/output parameter (inspected by the recipient of the message, and then altered).

WinRegisterUserMsg Parameter - datatype2

datatype2 (**LONG**) - input
Data type of message parameter 2.

See the description of *datatype1*.

WinRegisterUserMsg Parameter - dir2

dir2 (**LONG**) - input
Direction of message parameter 2.

See the description of *dir1*.

WinRegisterUserMsg Parameter - datatypeper

datatypeper (**LONG**) - input
Data type of message reply.

See the description of *datatype1*. The message reply is always an output parameter.

WinRegisterUserMsg Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinRegisterUserMsg - Parameters

hab (**HAB**) - input
Anchor-block handle.

msgid (**ULONG**) - input
Message identifier.

This must not be less than WM_USER, and must not have been defined previously.

datatype1 (**LONG**) - input
Data type of message parameter 1.

Valid data types are listed below. For data types that are shorter than 4 bytes, the data must be placed in the least-significant bytes, with the most significant bytes nullified (unsigned data types) or signed-extended (signed data types).

DTYP_BIT16	See BIT16 data type.
DTYP_BIT32	See BIT32 data type.
DTYP_BIT8	See BIT8 data type.
DTYP_BOOL	See BOOL data type.
DTYP_LONG	See LONG data type.
DTYP_SHORT	See SHORT data type.
DTYP_UCHAR	See UCHAR data type.
DTYP_ULONG	See ULONG data type.
DTYP_USHORT	See USHORT data type.
DTYP_P*	A pointer to a system data type. Note that not all of the system data types that exist in the CPI are valid.
< -DTYP_USER	A pointer to a user data type. The user data type must have already been defined via WinRegisterUserDatatype .

dir1 ([LONG](#)) - input

Direction of message parameter 1.

If the message parameter is a pointer, the direction values listed below apply to the *contents* of the storage location pointed at, as well as to the message parameter itself.

RUM_IN	Input parameter (inspected by the recipient of the message, but not altered)
RUM_OUT	Output parameter (altered by the recipient of the message, without inspecting its value first)
RUM_INOUT	Input/output parameter (inspected by the recipient of the message, and then altered).

datatype2 ([LONG](#)) - input

Data type of message parameter 2.

See the description of *datatype1*.

dir2 ([LONG](#)) - input

Direction of message parameter 2.

See the description of *dir1*.

datatypeper ([LONG](#)) - input

Data type of message reply.

See the description of *datatype1*. The message reply is always an output parameter.

rc ([BOOL](#)) - returns

Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinRegisterUserMsg - Remarks

This function has no effect unless the [RegisterUserHook](#) hook, which is invoked by this function, has been set.

It is an error to attempt to register the same message identifier more than once within a single OS/2 process.

WinRegisterUserMsg - Errors

Possible returns from [WinGetLastError](#)

PMERR_MSGID_TOO_SMALL (0x104F)
The message identifier specified is too small.

PMERR_DATATYPE_INVALID (0x1045)
An invalid datatype was specified.

PMERR_DATATYPE_TOO_LONG (0x1047)
The datatype specified was too long.

WinRegisterUserMsg - Related Functions

Related Functions

- [WinBroadcastMsg](#)
 - [WinCreateMsgQueue](#)
 - [WinDestroyMsgQueue](#)
 - [WinDispatchMsg](#)
 - [WinGetDlgMsg](#)
 - [WinGetMsg](#)
 - [WinInSendMsg](#)
 - [WinPeekMsg](#)
 - [WinPostMsg](#)
 - [WinPostQueueMsg](#)
 - [WinQueryMsgPos](#)
 - [WinQueryMsgTime](#)
 - [WinQueryQueueInfo](#)
 - [WinQueryQueueStatus](#)
 - [WinSendDlgItemMsg](#)
 - [WinSendMsg](#)
 - [WinSetClassMsgInterest](#)
 - [WinSetMsgInterest](#)
 - [WinSetMsgMode](#)
 - [WinSetSynchroMode](#)
 - [WinWaitMsg](#)
-

WinRegisterUserMsg - Example Code

This example uses the WinRegisterUserMsg call to register a user-defined message and define its parameters.

```
#define INCL_WINMESSAGEGR
```

```

#define INCL_WINTYPES
#include <OS2.H>
#define WM_MY_MESSAGE WM_USER + 11

HAB hab;

WinRegisterUserMessage(hab,
    WM_MY_MESSAGE,
    DTYP_BIT16, /* param1 is a USHORT */
    RUM_INOUT, /* param1 is input/output */
    DTYP_BIT16, /* param2 is a USHORT */
    RUM_INOUT, /* param2 is input/output */
    DTYP_BIT16); /* reply is a USHORT */

```

WinRegisterUserMsg - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinReleaseHook

WinReleaseHook - Syntax

This function releases an application hook from a hook chain.

```

#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
HMQ      hmq;          /* Handle of message queue from which the hook is to be released. */
LONG     lHook;        /* Type of hook chain. */
PFN      pAddress;     /* Address of the hook routine. */
HMODULE  Module;       /* Module handle. */
BOOL     rc;           /* Success indicator. */

rc = WinReleaseHook(hab, hmq, lHook, pAddress,
    Module);

```

WinReleaseHook Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinReleaseHook Parameter - hmq

hmq ([HMQ](#)) - input
Handle of message queue from which the hook is to be released.

HMQ_CURRENT
The hook is released from the message queue associated with the current thread (calling thread).

NULLHANDLE
The hook is released from the system hook chain.

WinReleaseHook Parameter - IHook

IHook ([LONG](#)) - input
Type of hook chain.

This parameter can have one of the following HK_* values:

HK_CHECKMSGFILTER
See [CheckMsgFilterHook](#).

HK_CODEPAGECHANGED
See [CodePageChangedHook](#).

HK_DESTROYWINDOW
See [DestroyWindowHook](#).

HK_FINDWORD
See [FindWordHook](#).

HK_FLUSHBUF
See [FlushBufHook](#).

HK_HELP
See [HelpHook](#).

HK_INPUT
See [InputHook](#).

HK_JOURNALPLAYBACK
See [JournalPlaybackHook](#).

HK_JOURNALRECORD
See [JournalRecordHook](#).

HK_LOADER
See [LoaderHook](#).

HK_LOCKUP
See [LockupHook](#).

HK_MSGCONTROL
See [MsgControlHook](#).

HK_MSGFILTER
See [MsgFilterHook](#).

HK_MSGINPUT
See [MsgInputHook](#).

HK_PLIST_ENTRY
See [ProgramListEntryHook](#).

HK_PLIST_EXIT
See [ProgramListExitHook](#).

HK_REGISTERUSERMSG
See [RegisterUserHook](#).

HK_SENDMSG
See [SendMsgHook](#).

HK_WINDOWDC

See [WindowDCHook](#)

WinReleaseHook Parameter - pAddress

pAddress ([PFN](#)) - input

Address of the hook routine.

WinReleaseHook Parameter - Module

Module ([HMODULE](#)) - input

Module handle.

NULLHANDLE

The hook procedure is in the application's .EXE file.

Module

This is the module that contains the application procedure, as returned by the DosLoadModule or DosQueryModuleHandle call.

WinReleaseHook Return Value - rc

rc ([BOOL](#)) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinReleaseHook - Parameters

hab ([HAB](#)) - input

Anchor-block handle.

hmq ([HMQ](#)) - input

Handle of message queue from which the hook is to be released.

HMQ_CURRENT

The hook is released from the message queue associated with the current thread (calling thread).

NULLHANDLE

The hook is released from the system hook chain.

IHook ([LONG](#)) - input

Type of hook chain.

This parameter can have one of the following HK_* values:

HK_CHECKMSGFILTER	See CheckMsgFilterHook .
HK_CODEPAGECHANGED	See CodePageChangedHook .
HK_DESTROYWINDOW	See DestroyWindowHook .
HK_FINDWORD	See FindWordHook .
HK_FLUSHBUF	See FlushBufHook .
HK_HELP	See HelpHook .
HK_INPUT	See InputHook .
HK_JOURNALPLAYBACK	See JournalPlaybackHook .
HK_JOURNALRECORD	See JournalRecordHook .
HK_LOADER	See LoaderHook .
HK_LOCKUP	See LockupHook .
HK_MSGCONTROL	See MsgControlHook .
HK_MSGFILTER	See MsgFilterHook .
HK_MSGINPUT	See MsgInputHook .
HK_PLIST_ENTRY	See ProgramListEntryHook .
HK_PLIST_EXIT	See ProgramListExitHook .
HK_REGISTERUSERMSG	See RegisterUserHook .
HK_SENDMSG	See SendMsgHook .
HK_WINDOWDC	See WindowDCHook .

pAddress ([PFN](#)) - input
Address of the hook routine.

Module ([HMODULE](#)) - input
Module handle.

NULLHANDLE	The hook procedure is in the application's .EXE file.
Module	This is the module that contains the application procedure, as returned by the DosLoadModule or DosQueryModuleHandle call.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinReleaseHook - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HMQ (0x1002)
An invalid message-queue handle was specified.

PMERR_PARAMETER_OUT_OF_RANGE (0x1003)
The value of a parameter was not within the defined valid range for that parameter.

WinReleaseHook - Related Functions

Related Functions

- [WinCallMsgFilter](#)
- [WinReleaseHook](#)
- [WinSetHook](#)

WinReleaseHook - Example Code

This example uses the WinReleaseHook call to release a hook that records user-input messages from the application queue.

```
#define INCL_WINHOOKS
#include <OS2.H>

void RecordHook(HAB hab, PQMSG pqmsg);

samp()
{
    HAB hab;

    WinSetHook(hab,
               HMQ_CURRENT,
               HK_JOURNALRECORD,
               (PFN)RecordHook,
               (HMODULE)0); /* hook is into application queue. */

    WinReleaseHook(hab,
                   HMQ_CURRENT,
                   HK_JOURNALRECORD,
                   (PFN)RecordHook,
                   (HMODULE)0); /* hook is into application queue, */

}

/* This hook records user-input messages. */
void RecordHook(HAB hab, PQMSG pqmsg)
{
    /* ... */
}
```

WinReleaseHook - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)

WinReleasePS

WinReleasePS - Syntax

This method releases a cache presentation space obtained using the [WinGetPS](#), the [WinGetScreenPS](#), or the [WinGetClipPS](#) call.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HPS      hps; /* Handle of the cache presentation space to release, as returned by the WinGetPS, the WinGetScr
BOOL      rc; /* Success indicator. */

rc = WinReleasePS(hps);
```

WinReleasePS Parameter - hps

hps ([HPS](#)) - input
Handle of the cache presentation space to release, as returned by the [WinGetPS](#), the [WinGetScreenPS](#), or the [WinGetClipPS](#) function.

WinReleasePS Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

WinReleasePS - Parameters

hps ([HPS](#)) - input

Handle of the cache presentation space to release, as returned by the [WinGetPS](#), the [WinGetScreenPS](#), or the [WinGetClipPS](#) function.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

WinReleasePS - Remarks

Only cache presentation spaces can be released using this method, after which the presentation space is returned to the cache to be used again. The presentation-space handle should not be used following this call.

WinReleasePS - Related Functions

Related Functions

- [WinBeginPaint](#)
- [WinEnableWindowUpdate](#)
- [WinEndPaint](#)
- [WinExcludeUpdateRegion](#)
- [WinGetClipPS](#)
- [WinGetPS](#)
- [WinGetScreenPS](#)
- [WinInvalidateRect](#)
- [WinInvalidateRegion](#)
- [WinIsWindowShowing](#)
- [WinIsWindowVisible](#)
- [WinLockVisRegions](#)
- [WinOpenWindowDC](#)
- [WinQueryUpdateRect](#)
- [WinQueryUpdateRegion](#)
- [WinRealizePalette](#)
- [WinReleasePS](#)
- [WinShowWindow](#)
- [WinUpdateWindow](#)
- [WinValidateRect](#)
- [WinValidateRegion](#)

WinReleasePS - Example Code

This example shows how a thread can access a presentation space, draw to it, and release it.

```
#define INCL_DOSSEMAPHORES
#define INCL_GPIPRIMITIVES
#define INCL_WINWINDOWMGR
#include <OS2.H>
HPS hps;

hps = WinGetPS( hwndClient );

/* Draw client area */
```



```

        .
        .
        .
        */
        */

/* Release the presentation space */
WinReleasePS( hps );

```

WinReleasePS - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinRemovePresParam

WinRemovePresParam - Syntax

This function removes a presentation parameter associated with the window *hwnd*.

```

#define INCL_WINSYS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd;          /* Window handle whose presentation is to be change. */
ULONG    idAttrType;    /* Type of presentation parameter attribute to be removed. */
BOOL     rc;            /* Success indicator. */

rc = WinRemovePresParam(hwnd, idAttrType);

```

WinRemovePresParam Parameter - hwnd

hwnd (**HWND**) - input
 Window handle whose presentation is to be change.

WinRemovePresParam Parameter - idAttrType

idAttrType (ULONG) - input

Type of presentation parameter attribute to be removed.

This parameter can be set to one of the following system-defined presentation parameter attribute types or an application-defined type.

PP_FOREGROUND_COLOR

Foreground color (in RGB) attribute.

PP_BACKGROUND_COLOR

Background color (in RGB) attribute.

PP_FOREGROUND_COLOR_INDEX

Foreground color index attribute.

PP_BACKGROUND_COLOR_INDEX

Background color index attribute.

PP_HILITE_FOREGROUND_COLOR

Highlighted foreground color (in RGB) attribute, for example for selected menu items.

PP_HILITE_BACKGROUND_COLOR

Highlighted background color (in RGB) attribute.

PP_HILITE_FOREGROUND_COLOR_INDEX

Highlighted foreground color index attribute.

PP_HILITE_BACKGROUND_COLOR_INDEX

Highlighted background color index attribute.

PP_DISABLED_FOREGROUND_COLOR

Disabled foreground color (in RGB) attribute.

PP_DISABLED_BACKGROUND_COLOR

Disabled background color (in RGB) attribute.

PP_DISABLED_FOREGROUND_COLOR_INDEX

Disabled foreground color index attribute.

PP_DISABLED_BACKGROUND_COLOR_INDEX

Disabled background color index attribute.

PP_BORDER_COLOR

Border color (in RGB) attribute.

PP_BORDER_COLOR_INDEX

Border color index attribute.

PP_FONT_NAME_SIZE

Font name and size attribute.

PP_ACTIVE_COLOR

Active color value of data type RGB.

PP_ACTIVE_COLOR_INDEX

Active color index value of data type LONG.

PP_INACTIVE_COLOR

Inactive color value of data type RGB.

PP_INACTIVE_COLOR_INDEX

Inactive color index value of data type LONG.

PP_ACTIVE_TEXT_FGND_COLOR

Active text foreground color value of data type RGB.

PP_ACTIVE_TEXT_FGND_COLOR_INDEX

Active text foreground color index value of data type LONG.

PP_ACTIVE_TEXT_BGND_COLOR

Active text background color value of data type [RGB](#).

PP_ACTIVETEXTBGNDINDEX
Active text background color index value of data type [LONG](#).

PP_INACTIVETEXTFGNDCOLOR
Inactive text foreground color value of data type [RGB](#).

PP_INACTIVETEXTFGNDINDEX
Inactive text foreground color index value of data type [LONG](#).

PP_INACTIVETEXTBGNDCOLOR
Inactive text background color value of data type [RGB](#).

PP_INACTIVETEXTBGNDINDEX
Inactive text background color index value of data type [LONG](#).

PP_SHADOW
Changes the color used for drop shadows on certain controls.

PP_USER
This is a user-defined presentation parameter.

WinRemovePresParam Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE
Successful completion

FALSE
Error occurred.

WinRemovePresParam - Parameters

hwnd ([HWND](#)) - input
Window handle whose presentation is to be change.

idAttrType ([ULONG](#)) - input
Type of presentation parameter attribute to be removed.

This parameter can be set to one of the following system-defined presentation parameter attribute types or an application-defined type.

PP_FOREGROUND
Foreground color (in [RGB](#)) attribute.

PP_BACKGROUND
Background color (in [RGB](#)) attribute.

PP_FOREGROUNDINDEX
Foreground color index attribute.

PP_BACKGROUNDINDEX
Background color index attribute.

PP_HILITEFOREGROUND
Highlighted foreground color (in [RGB](#)) attribute, for example for selected menu items.

PP_HILITEBACKGROUNDCOLOR
Highlighted background color (in [RGB](#)) attribute.

PP_HILITEFOREGROUNDCOLORINDEX
Highlighted foreground color index attribute.

PP_HILITEBACKGROUNDINDEX
Highlighted background color index attribute.

PP_DISABLEDFOREGROUNDCOLOR
Disabled foreground color (in [RGB](#)) attribute.

PP_DISABLEDBACKGROUNDINDEX
Disabled background color (in [RGB](#)) attribute.

PP_DISABLEDFOREGROUNDCOLORINDEX
Disabled foreground color index attribute.

PP_DISABLEDBACKGROUNDINDEX
Disabled background color index attribute.

PP_BORDERCOLOR
Border color (in [RGB](#)) attribute.

PP_BORDERINDEX
Border color index attribute.

PP_FONTNAME
Font name and size attribute.

PP_ACTIVECOLOR
Active color value of data type [RGB](#).

PP_ACTIVEINDEX
Active color index value of data type [LONG](#).

PP_INACTIVECOLOR
Inactive color value of data type [RGB](#).

PP_INACTIVEINDEX
Inactive color index value of data type [LONG](#).

PP_ACTIVETEXTFGNDINDEX
Active text foreground color value of data type [RGB](#).

PP_ACTIVETEXTFGNDINDEX
Active text foreground color index value of data type [LONG](#).

PP_ACTIVETEXTBGNDINDEX
Active text background color value of data type [RGB](#).

PP_ACTIVETEXTBGNDINDEX
Active text background color index value of data type [LONG](#).

PP_INACTIVETEXTFGNDINDEX
Inactive text foreground color value of data type [RGB](#).

PP_INACTIVETEXTFGNDINDEX
Inactive text foreground color index value of data type [LONG](#).

PP_INACTIVETEXTBGNDINDEX
Inactive text background color value of data type [RGB](#).

PP_INACTIVETEXTBGNDINDEX
Inactive text background color index value of data type [LONG](#).

PP_SHADOW
Changes the color used for drop shadows on certain controls.

PP_USER
This is a user-defined presentation parameter.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinRemovePresParam - Remarks

When a presentation parameter is removed, a [WM_PRESPARAMCHANGED](#) message is sent to all windows owned by the window calling the [WinSetPresParam](#) function. (See also [WinSetPresParam](#) and [WinQueryPresParam](#).)

WinRemovePresParam - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinRemovePresParam - Related Functions

Related Functions

- [WinDrawBitmap](#)
- [WinDrawBorder](#)
- [WinDrawPointer](#)
- [WinDrawText](#)
- [WinFillRect](#)
- [WinGetSysBitmap](#)
- [WinInvertRect](#)
- [WinQueryPresParam](#)
- [WinRemovePresParam](#)
- [WinScrollWindow](#)
- [WinSetPresParam](#)

WinRemovePresParam - Example Code

This example removes the disable-foreground attribute after querying to ensure that the referenced window has this attribute defined.

```
#define INCL_WINSYS                /* System values                */
#include <os2.h>

HWND  hwnd;                        /* window handle                */
ULONG AttrFound;                   /* attributes found              */
ULONG AttrValue[32];              /* attribute value               */
ULONG cbRetLen;                   /* length of returned value     */

WinRemovePresParam(hwnd, PP_DISABLEDFOREGROUNDINDEX);
```

WinRemovePresParam - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinRemoveSwitchEntry

WinRemoveSwitchEntry - Syntax

This function removes a specified entry from the Window List.

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HSWITCH    hswitchSwitch; /* Switch-list (Window List) entry handle. */
ULONG      usRetCode;     /* Success indicator. */

usRetCode = WinRemoveSwitchEntry(hswitchSwitch);
```

WinRemoveSwitchEntry Parameter - hswitchSwitch

hswitchSwitch ([HSWITCH](#)) - input
Switch-list (Window List) entry handle.

WinRemoveSwitchEntry Return Value - usRetCode

usRetCode ([ULONG](#)) - returns
Success indicator.

0	Successful completion
Other	Error occurred.

WinRemoveSwitchEntry - Parameters

hswitchSwitch ([HSWITCH](#)) - input
Switch-list (Window List) entry handle.

usRetCode ([ULONG](#)) - returns
Success indicator.

0	Successful completion
Other	Error occurred.

WinRemoveSwitchEntry - Remarks

An application that uses the operating system effectively should, at least, add its main window to the Window List when it starts, and remove it from the Window List when it stops.

Window List entries for non-OS/2 applications cannot be removed using this function. These entries are removed automatically by the system when the session they occupy terminates.

Note: This function and the [WinCreateSwitchEntry](#) and [WinAddSwitchEntry](#) functions are not required if the main window is created with the frame style FCF_TASKLIST or FCF_STANDARD, as these styles automatically update the Window List when the main window is created, destroyed, or its title changes.

WinRemoveSwitchEntry - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_SWITCH_HANDLE (0x1202)
An invalid Window List entry handle was specified.

PMERR_INVALID_WINDOW (0x1206)
The window specified with a Window List call is not a valid frame window.

WinRemoveSwitchEntry - Related Functions

Related Functions

- [WinAddSwitchEntry](#)

- [WinChangeSwitchEntry](#)
- [WinCreateSwitchEntry](#)
- [WinQuerySessionTitle](#)
- [WinQuerySwitchEntry](#)
- [WinQuerySwitchHandle](#)
- [WinQuerySwitchList](#)
- [WinQueryTaskSizePos](#)
- [WinQueryTaskTitle](#)
- [WinRemoveSwitchEntry](#)
- [WinSwitchToProgram](#)

WinRemoveSwitchEntry - Example Code

This example calls WinQuerySwitchHandle to get the Task List handle of a frame window, and then calls WinRemoveSwitchEntry to remove it.

```
#define INCL_WINSWITCHLIST
#include <OS2.H>
HAB hab;
HWND hwndFrame;
HSWITCH hswitch;
SWCNTRL swctl;

hswitch = WinQuerySwitchHandle(hwndFrame, 0);
WinRemoveSwitchEntry(hswitch);
```

WinRemoveSwitchEntry - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Errors](#)
- [Remarks](#)
- [Example Code](#)
- [Related Functions](#)
- [Glossary](#)

WinReplaceObjectClass

WinReplaceObjectClass - Syntax

The WinReplaceObjectClass function replaces a registered class with another registered class. If *fReplace* is FALSE, *pOldClassName* will revert back to its original definition.


```

#define INCL_WINWORKPLACE
#include <os2.h>

PSZ      pOldClassName; /* Pointer to class name. */
PSZ      pNewClassName; /* Pointer to new class name. */
BOOL     fReplace;      /* Function replacement flag. */
BOOL     rc;             /* Success indicator. */

rc = WinReplaceObjectClass(pOldClassName,
    pNewClassName, fReplace);

```

WinReplaceObjectClass Parameter - pOldClassName

pOldClassName (**PSZ**) - input
 Pointer to class name.

A pointer to a zero-terminated string which contains the name of the object class being replaced by *pNewClassName* in the workplace.

WinReplaceObjectClass Parameter - pNewClassName

pNewClassName (**PSZ**) - input
 Pointer to new class name.

A pointer to a zero-terminated string which contains the name of the object class replacing the *pOldClassName* class.

WinReplaceObjectClass Parameter - fReplace

fReplace (**BOOL**) - input
 Function replacement flag.

TRUE
 Replace the function of class *pOldClassName* with the function of the class *pNewClassName*.

FALSE
 Undo the replacement of the *pOldClassName* with *pNewClassName* by restoring the *pOldClassName* back to its original functionality.

WinReplaceObjectClass Return Value - rc

rc (**BOOL**) - returns
 Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinReplaceObjectClass - Parameters

pOldClassName ([PSZ](#)) - input
Pointer to class name.

A pointer to a zero-terminated string which contains the name of the object class being replaced by *pNewClassName* in the workplace.

pNewClassName ([PSZ](#)) - input
Pointer to new class name.

A pointer to a zero-terminated string which contains the name of the object class replacing the *pOldClassName* class.

fReplace ([BOOL](#)) - input
Function replacement flag.

TRUE	Replace the function of class <i>pOldClassName</i> with the function of the class <i>pNewClassName</i> .
FALSE	Undo the replacement of the <i>pOldClassName</i> with <i>pNewClassName</i> by restoring the <i>pOldClassName</i> back to its original functionality.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinReplaceObjectClass - Remarks

The class specified by *pNewClassName* must be a descendant of the class specified by *pOldClassName*, otherwise an error will be returned. Replacing an object is useful if it is desired to modify the behavior of objects which are instances of the class *pOldClassName* and which are not aware of the class *pNewClassName*.

WinReplaceObjectClass - Related Functions

Related Functions

- WinCreateObject
- [WinDeregisterObjectClass](#)
- [WinRegisterObjectClass](#)

WinReplaceObjectClass - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Related Functions](#)

[Glossary](#)

WinRequestMutexSem

WinRequestMutexSem - Syntax

WinRequestMutexSem requests ownership of a mutex semaphore or waits for a Presentation Manager message.

```
#define INCL_WINMESSAGEGR
#include <os2.h>

HMTX      hmtx;          /* The handle of the mutex semaphore to request. */
ULONG     ulTimeout;     /* Time-out in milliseconds. */
APIRET    ulrc;          /* Return Code. */

ulrc = WinRequestMutexSem(hmtx, ulTimeout);
```

WinRequestMutexSem Parameter - hmtx

hmtx (**HMTX**) - input

The handle of the mutex semaphore to request.

WinRequestMutexSem Parameter - ulTimeout

ulTimeout (**ULONG**) - input

Time-out in milliseconds.

This is the maximum amount of time the user wants to allow the thread to be blocked.

This parameter can also have the following values:

SEM_IMMEDIATE_RETURN (0)

WinRequestMutexSem returns immediately without blocking the calling thread.

SEM_INDEFINITE_WAIT (-1)

WinRequestMutexSem blocks the calling thread indefinitely.

WinRequestMutexSem Return Value - ulrc

ulrc ([APIRET](#)) - returns
Return Code.

WinRequestMutexSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
95	ERROR_INTERRUPT
103	ERROR_TOO_MANY_SEM_REQUESTS
105	ERROR_SEM_OWNER_DIED
640	ERROR_TIMEOUT

WinRequestMutexSem - Parameters

hmtx ([HMTX](#)) - input
The handle of the mutex semaphore to request.

ulTimeout ([ULONG](#)) - input
Time-out in milliseconds.

This is the maximum amount of time the user wants to allow the thread to be blocked.

This parameter can also have the following values:

SEM_IMMEDIATE_RETURN (0)	WinRequestMutexSem returns immediately without blocking the calling thread.
SEM_INDEFINITE_WAIT (-1)	WinRequestMutexSem blocks the calling thread indefinitely.

ulrc ([APIRET](#)) - returns
Return Code.

WinRequestMutexSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
95	ERROR_INTERRUPT
103	ERROR_TOO_MANY_SEM_REQUESTS
105	ERROR_SEM_OWNER_DIED
640	ERROR_TIMEOUT

WinRequestMutexSem - Remarks

WinRequestMutexSem is similar to DosRequestMutexSem and requests ownership of a mutex semaphore or waits for a window message sent by the WinSendMsg function from another thread to be received.

This function can be called by any thread in the process that created the semaphore. Threads in other processes can also call this function, but they must first gain access to the semaphore by issuing DosOpenMutexSem.

Since the processing of a window message may take longer than the value specified by the *ulTimeout* parameter, this function may not return within the time specified by that value.

WinRequestMutexSem - Related Functions

Related Functions

- [WinSendMsg](#)
- [WinPostMsg](#)

WinRequestMutexSem - Example Code

This example requests ownership of a mutex semaphore. Assume that the handle of the semaphore has been placed into *hmtx* already.

ulTimeout is the number of milliseconds that the calling thread will wait for ownership of the mutex semaphore. If the specified mutex semaphore is not released during this time interval, the calling thread does not receive ownership of it.

```
#define INCL_DOSSEMAPHORES /* Semaphore values */
#define INCL_WINMESSAGEGR

#include <os2.h>
#include <stdio.h>

#ifndef ERROR_TIMEOUT
#define ERROR_TIMEOUT 640
#define ERROR_INTERRUPT 95
#endif

HMTX hmtx; /* Mutex semaphore handle */
ULONG ulTimeout; /* Number of milliseconds to wait */
ULONG rc; /* Return code */

ulTimeout = 60000; /* Wait for a maximum of 1 minute */

rc = WinRequestMutexSem(hmtx, ulTimeout);

if (rc == ERROR_TIMEOUT)
{
    printf("WinRequestMutexSem call timed out");
    return;
}

if (rc == ERROR_INTERRUPT)
{
    printf("WinRequestMutexSem call was interrupted");
    return;
}

if (rc != 0)
{
    printf("WinRequestMutexSem error: return code = %ld", rc);
    return;
}
```

WinRequestMutexSem - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinRestartSOMDD

WinRestartSOMDD - Syntax

This function starts the DSOM daemon.

```
#define INCL_WPCLASS
#include <os2.h>

BOOL      fAction; /* Flag indicating action to perform on the DSOM daemon. */
APIRET    rc;      /* Return code. */

rc = WinRestartSOMDD(fAction);
```

WinRestartSOMDD Parameter - fAction

fAction ([BOOL](#)) - input
Flag indicating action to perform on the DSOM daemon.

TRUE	Start the DSOM daemon.
FALSE	Stop the DSOM daemon.

WinRestartSOMDD Return Value - rc

rc ([APIRET](#)) - returns
Return code.

PMERR_OK (0x0000)	Successfully invoked.
PMERR_WPDSEVER_IS_ACTIVE (0x1056)	The Workplace Shell DSOM Server is active and running. It cannot stop the DSOM daemon until the Workplace Shell DSOM Server has stopped.

PMERR_SOMDD_IS_ACTIVE (0x1058)

The DSOM daemon has been started by the Workplace Shell process and is active and running; therefore, it cannot be started again.

PMERR_SOMDD_NOT_STARTED (0x1059)

The DSOM daemon failed to start. It might be active in another process.

WinRestartSOMDD - Parameters

fAction ([BOOL](#)) - input

Flag indicating action to perform on the DSOM daemon.

TRUE

Start the DSOM daemon.

FALSE

Stop the DSOM daemon.

rc ([APIRET](#)) - returns

Return code.

PMERR_OK (0x0000)

Successfully invoked.

PMERR_WPDSEVER_IS_ACTIVE (0x1056)

The Workplace Shell DSOM Server is active and running. It cannot stop the DSOM daemon until the Workplace Shell DSOM Server has stopped.

PMERR_SOMDD_IS_ACTIVE (0x1058)

The DSOM daemon has been started by the Workplace Shell process and is active and running; therefore, it cannot be started again.

PMERR_SOMDD_NOT_STARTED (0x1059)

The DSOM daemon failed to start. It might be active in another process.

WinRestartSOMDD - Remarks

This function starts the DSOM daemon in the Workplace Shell process as a background process invisible to the users. The status of the daemon can be determined at any time by calling [WinIsSOMDDReady](#).

Note: This function requires that the PM Shell is up and running.

WinRestartSOMDD - Related Functions

Related Functions

- [WinIsSOMDDReady](#)
- [WinIsWPDServerReady](#)
- [WinRestartWPDServer](#)

WinRestartSOMDD - Example Code

This example starts the DSOM daemon within the Workplace Shell process.

```
#define INCL_WPCLASS
#include <os2.h>

enum {OFF, ON};
APIRET apiRtnCd;

apiRtnCd = WinRestartSOMDD(ON);
if (apiRtnCd == PMERR_SOMDD_NOT_STARTED)
    somPrintf ("DSOM Daemon failed to start; might be active already");
```

WinRestartSOMDD - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinRestartWPDServer

WinRestartWPDServer - Syntax

This function starts the Workplace Shell DSOM server.

```
#define INCL_WPCLASS
#include <os2.h>

BOOL      fAction; /* Flag indicating the action to perform on the Workplace Shell DSOM server. */
APIRET    rc;      /* Return code. */

rc = WinRestartWPDServer(fAction);
```

WinRestartWPDServer Parameter - fAction

fAction (BOOL) - input

Flag indicating the action to perform on the Workplace Shell DSOM server.

TRUE

Start the Workplace Shell DSOM server.

FALSE

Stop the Workplace Shell DSOM server.

WinRestartWPDServer Return Value - rc

rc (APIRET) - returns

Return code.

PMERR_OK (0x0000)

Successfully invoked.

PMERR_WPDSEVER_IS_ACTIVE (0x1056)

The Workplace Shell DSOM server is active and cannot be started again.

PMERR_WPDSEVER_NOT_STARTED (0x1057)

The Workplace Shell DSOM server could not be started, possibly related to a corrupted copy of WPDSRVP.DLL.

Other

Any other non-zero value could indicate that the WPDSRVP.DLL is corrupted or not found.

WinRestartWPDServer - Parameters

fAction (BOOL) - input

Flag indicating the action to perform on the Workplace Shell DSOM server.

TRUE

Start the Workplace Shell DSOM server.

FALSE

Stop the Workplace Shell DSOM server.

rc (APIRET) - returns

Return code.

PMERR_OK (0x0000)

Successfully invoked.

PMERR_WPDSEVER_IS_ACTIVE (0x1056)

The Workplace Shell DSOM server is active and cannot be started again.

PMERR_WPDSEVER_NOT_STARTED (0x1057)

The Workplace Shell DSOM server could not be started, possibly related to a corrupted copy of WPDSRVP.DLL.

Other

Any other non-zero value could indicate that the WPDSRVP.DLL is corrupted or not found.

WinRestartWPDServer - Remarks

This function requires that the DSOM daemon is started first and must be running successfully in order for the Workplace Shell DSOM

server to become ready. The status of the Workplace Shell DSOM server can be obtained at any time using [WinIsWPDServerReady](#). Once the DSOM server is ready, a client Workplace Shell DSOM server application can be executed.

Note: This function requires that the PM Shell is up and running.

WinRestartWPDServer - Related Functions

Related Functions

- [WinIsSOMDDReady](#)
 - [WinIsWPDServerReady](#)
 - [WinRestartSOMDD](#)
-

WinRestartWPDServer - Example Code

This example starts the DSOM daemon within the Workplace Shell process and then starts the Workplace Shell DSOM server.

```
#define INCL_WINERRORS
#define INCL_WPCCLASS
#include <os2.h>

enum      {OFF, ON};
APIRET    apiRtnCd;

apiRtnCd=WinRestartSOMDD(ON);
if (apiRtnCd == PMERR_OK)
{
    apiRtnCd = WinRestartWPDServer(ON);
    if (apiRtnCd)
    {
        PERRINFO perriErrorInfo;

        if (apiRtnCd == PMERR_WPDSEVER_IS_ACTIVE)
        {
            somPrintf("The Workplace Shell DSOM server is already running");
            exit(2);
        }
        if (apiRtnCd == PMERR_WPDSEVER_NOT_STARTED)
        {
            somPrintf("Failed to start the Workplace Shell DSOM server");
            exit(2);
        }

        /* Obtain the error block */
        /* WinGetErrorInfo is for DosLoadModule related errors only */
        perriErrorInfo = WinGetErrorInfo(hab);
        somPrintf("Loading WPDSEVER.DLL failed: %ld", perriErrorInfo->idError);
        WinFreeErrorInfo(perriErrorInfo);
        exit(99);
    }
    somPrintf("All's well!");
}
else
{
    somPrintf("Failed to start the DSOM daemon");
    exit(1);
}
```

WinRestartWPDServer - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinRestoreWindowPos

WinRestoreWindowPos - Syntax

The WinRestoreWindowPos function will restore the size and position of the window specified by *hwnd* to the state it was in when [WinStoreWindowPos](#) was last called with the same *pAppName* and *pKeyName*.

```
#define INCL_WINWORKPLACE
#include <os2.h>

PSZ      pAppName; /* Pointer to application name. */
PSZ      pKeyName; /* Pointer to key name. */
HWND     hwnd;     /* Window handle of the window to restore. */
BOOL     rc;       /* Success indicator. */

rc = WinRestoreWindowPos(pAppName, pKeyName,
                        hwnd);
```

WinRestoreWindowPos Parameter - pAppName

pAppName ([PSZ](#)) - input

Pointer to application name.

A pointer to a zero-terminated string which contains the application name.

WinRestoreWindowPos Parameter - pKeyName

pKeyName ([PSZ](#)) - input

Pointer to key name.

A pointer to a zero-terminated string which contains the key name.

WinRestoreWindowPos Parameter - hwnd

hwnd ([HWND](#)) - input
Window handle of the window to restore.

WinRestoreWindowPos Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinRestoreWindowPos - Parameters

pAppName ([PSZ](#)) - input
Pointer to application name.

A pointer to a zero-terminated string which contains the application name.

pKeyName ([PSZ](#)) - input
Pointer to key name.

A pointer to a zero-terminated string which contains the key name.

hwnd ([HWND](#)) - input
Window handle of the window to restore.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinRestoreWindowPos - Remarks

This function will also restore presentation parameters which were saved by a previous call to [WinStoreWindowPos](#).

WinRestoreWindowPos - Related Functions

Related Functions

- [WinStoreWindowPos](#)
-

WinRestoreWindowPos - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

WinSaveObject

WinSaveObject - Syntax

This function is specific to OS/2 Version 3.0 or higher.

This function saves the state of an object.

```
#define INCL_WINWORKPLACE
#include <os2.h>

HOBJECT    hObject; /* Handle of the object to be saved. */
BOOL       fAsync;  /* Asynchronous flag. */
BOOL       rc;      /* Success indicator. */

rc = WinSaveObject(hObject, fAsync);
```

WinSaveObject Parameter - hObject

hObject ([HOBJECT](#)) - input
Handle of the object to be saved.

WinSaveObject Parameter - fAsync

fAsync (BOOL) - input
Asynchronous flag.

TRUE	Calls asynchronous save.
FALSE	Calls synchronous save.

WinSaveObject Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

WinSaveObject - Parameters

hObject (HOBJECT) - input
Handle of the object to be saved.

fAsync (BOOL) - input
Asynchronous flag.

TRUE	Calls asynchronous save.
FALSE	Calls synchronous save.

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

WinSaveObject - Remarks

If an asynchronous save is called, the object will be saved on a separate thread ("lazy written"); this is the preferred method for saving. Otherwise, the object is saved on the user interface thread.

Using [HOBJECT](#) for .INI files or files in which an application uses a rename/save/delete sequence is not supported.

WinSaveObject - Related Functions

Related Functions

- [WinCopyObject](#)
 - [WinCreateObject](#)
 - [WinDestroyObject](#)
 - [WinMoveObject](#)
 - [WinQueryObjectWindow](#)
-

WinSaveObject - Example Code

```
#define INCL_WINWORKPLACE
#include "os2.h"

HOBJECT  Object;
BOOL     fAsync = TRUE;
BOOL     fSuccess

hObject = WinQueryObject("<WP_DESKTOP>");
if (hObject != NULL)
{
    /* WinQueryObject was successful */
    fSuccess = WinSaveObject(hObject,fAsync);
    if (fSuccess)
    {
        /* The state of the Desktop was saved Asynchronously */
    }
    else
    {
        /* Asynchronous save of the Desktop failed */
    }
}
```

WinSaveObject - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSaveWindowPos

WinSaveWindowPos - Syntax

This function associates an array of [SWP](#) structures with the process of repositioning a frame window.

```
#define INCL_WINFRAMEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HSAVEWP    hsvwp; /* Identifier of the frame window repositioning process. */
PSWP       pswp;  /* Array of SWP structures. */
ULONG      cswp;  /* Count of SWP structures. */
BOOL       rc;    /* Success indicator. */

rc = WinSaveWindowPos(hsvwp, pswp, cswp);
```

WinSaveWindowPos Parameter - hsvwp

hsvwp ([HSAVEWP](#)) - input
Identifier of the frame window repositioning process.

This handle is provided in the second parameter of the [WM_ADJUSTFRAMEPOS](#) message.

WinSaveWindowPos Parameter - pswp

pswp ([PSWP](#)) - input
Array of [SWP](#) structures.

WinSaveWindowPos Parameter - cswp

cswp ([ULONG](#)) - input
Count of [SWP](#) structures.

WinSaveWindowPos Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSaveWindowPos - Parameters

hsvwp ([HSAVEWP](#)) - input
Identifier of the frame window repositioning process.

This handle is provided in the second parameter of the [WM_ADJUSTFRAMEPOS](#) message.

pswp ([PSWP](#)) - input
Array of [SWP](#) structures.

cswp ([ULONG](#)) - input
Count of [SWP](#) structures.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSaveWindowPos - Remarks

This function is used only during the processing of the [WM_ADJUSTFRAMEPOS](#) message.

WinSaveWindowPos - Related Functions

Related Functions

- [WinGetMinPosition](#)
 - [WinQueryActiveWindow](#)
 - [WinQueryWindowPos](#)
 - [WinSaveWindowPos](#)
 - [WinSetActiveWindow](#)
 - [WinSetMultWindowPos](#)
 - [WinSetWindowPos](#)
-

WinSaveWindowPos - Related Messages

Related Messages

- [WM_ADJUSTFRAMEPOS](#)

WinSaveWindowPos - Example Code

This example shows how the repositioning of a window is recorded in SWP structures with the WinSaveWindowPos call.

```
#define INCL_WINFRAMEMGR
#include <OS2.H>
#define COUNT 10
HSAVEWP hsvwp;
SWP aswp[COUNT];
ULONG msg;

switch(msg){

case WM_ADJUSTFRAMEPOS:
WinSaveWindowPos(hsvwp,aswp,COUNT);
}
```

WinSaveWindowPos - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinScrollWindow

WinScrollWindow - Syntax

This function scrolls the contents of a window rectangle.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND      hwnd;          /* Window handle. */
LONG      lDx;            /* Amount of horizontal scroll to the right (in device units). */
LONG      lDy;            /* Amount of vertical scroll upward (in device units). */
PRECTL    prclScroll;     /* Scroll rectangle. */
PRECTL    prclClip;       /* Clip rectangle. */
```

```

HRGN      hrgnUpdateRgn; /* Update region. */
PRECTL    prclUpdate;    /* Update rectangle. */
ULONG     flOptions;      /* Scroll options. */
LONG      lComplexity;    /* Complexity of resulting region/error indicator. */

lComplexity = WinScrollWindow(hwnd, lDx, lDy,
                             prclScroll, prclClip, hrgnUpdateRgn,
                             prclUpdate, flOptions);

```

WinScrollWindow Parameter - hwnd

hwnd ([HWND](#)) - input
Window handle.

WinScrollWindow Parameter - lDx

lDx ([LONG](#)) - input
Amount of horizontal scroll to the right (in device units).

WinScrollWindow Parameter - lDy

lDy ([LONG](#)) - input
Amount of vertical scroll upward (in device units).

WinScrollWindow Parameter - prclScroll

prclScroll ([PRECTL](#)) - input
Scroll rectangle.

If this is NULL, the entire window is scrolled.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) may also be used, if supported by the language.

WinScrollWindow Parameter - prclClip

prclClip ([PRECTL](#)) - input
Clip rectangle.

If not NULL, this defines a clip rectangle that clips the destination of the scroll.

WinScrollWindow Parameter - hrgnUpdateRgn

hrgnUpdateRgn ([HRGN](#)) - input
Update region.

If not NULLHANDLE, this contains the region uncovered by the scroll when returned.

WinScrollWindow Parameter - prclUpdate

prclUpdate ([PRECTL](#)) - in/out
Update rectangle.

If not NULL, this contains the bounding rectangle of the invalid bits uncovered by the scroll when returned.

WinScrollWindow Parameter - flOptions

flOptions ([ULONG](#)) - input
Scroll options.

SW_SCROLLCHILDREN
Unless this is set, child windows are not scrolled. If this is set, and *prclScroll* is NULL, all the child windows are scrolled by *lDx* and *lDy* units. If *prclScroll* is not NULL, only those child windows that intersect *prclScroll* are scrolled.

SW_INVALIDATERGN
The invalid region created as a result of the scroll is added to the update regions of those windows affected. This may result in sending [WM_PAINT](#) messages to CS_SYNCPAINT windows before the call returns.

WinScrollWindow Return Value - lComplexity

lComplexity ([LONG](#)) - returns
Complexity of resulting region/error indicator.

RGN_NULL NULL rectangle invalid
RGN_RECT Simple rectangle invalid
RGN_COMPLEX Complex rectangle invalid

RGN_ERROR
Error.

WinScrollWindow - Parameters

hwnd ([HWND](#)) - input
Window handle.

IDx ([LONG](#)) - input
Amount of horizontal scroll to the right (in device units).

IDy ([LONG](#)) - input
Amount of vertical scroll upward (in device units).

prcIScroll ([PRECTL](#)) - input
Scroll rectangle.

If this is NULL, the entire window is scrolled.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) may also be used, if supported by the language.

prcIClip ([PRECTL](#)) - input
Clip rectangle.

If not NULL, this defines a clip rectangle that clips the destination of the scroll.

hrgnUpdateRgn ([HRGN](#)) - input
Update region.

If not NULLHANDLE, this contains the region uncovered by the scroll when returned.

prcIUpdate ([PRECTL](#)) - in/out
Update rectangle.

If not NULL, this contains the bounding rectangle of the invalid bits uncovered by the scroll when returned.

flOptions ([ULONG](#)) - input
Scroll options.

SW_SCROLLCHILDREN
Unless this is set, child windows are not scrolled. If this is set, and *prcIScroll* is NULL, all the child windows are scrolled by *IDx* and *IDy* units. If *prcIScroll* is not NULL, only those child windows that intersect *prcIScroll* are scrolled.

SW_INVALIDATERGN
The invalid region created as a result of the scroll is added to the update regions of those windows affected. This may result in sending [WM_PAINT](#) messages to CS_SYNCPAINT windows before the call returns.

lComplexity ([LONG](#)) - returns
Complexity of resulting region/error indicator.

RGN_NULL
NULL rectangle invalid
RGN_RECT
Simple rectangle invalid
RGN_COMPLEX
Complex rectangle invalid
RGN_ERROR
Error.

WinScrollWindow - Remarks

This function scrolls the contents of a rectangle defined by *prc/Scroll* in the window *hwnd*, by *IDx* units horizontally and *IDy* units vertically. All coordinates must be in device units.

Clipping takes place on the final image of the scrolling. Even if the scroll rectangle lies outside the clip rectangle, these bits are scrolled, if their destination lies within the intersection of the clip rectangle and the destination rectangle.

This function returns an RGN_* value, indicating the type of invalid region created by the scroll as returned by GpiCombineRegion. RGN_ERROR is returned if *hwnd* is invalid.

Note: If *hwnd* has style WS_CLIPCHILDREN, portions of any child window within the scroll area are scrolled. In this instance, this function must be called with *fOptions* SW_SCROLLCHILDREN.

This is the only function that can be used by a thread to move bits within its own window, because of the critical section nature of window update regions.

Scrolling is fastest without SW_SCROLLCHILDREN and SW_INVALIDATERGN. When scrolling needs to be repeated quickly, do not include the SW_INVALIDATERGN flag and repaint the invalid area if the invalid region is rectangular, otherwise invalidate and update.

If the scrolling is infrequent, include the SW_INVALIDATERGN flag. This function invalidates and updates synchronous-paint windows automatically before returning.

The cursor and the track rectangle are scrolled when they intersect with the scrolled region. Any part of the window's initial update region that intersects the scrolled region is also offset.

WinScrollWindow - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_INVALID_FLAG (0x1019)
An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

PMERR_HRGN_BUSY (0x2034)
An internal region busy error was detected. The region was locked by one thread during an attempt to access it from another thread.

WinScrollWindow - Related Functions

Related Functions

- [WinDrawBitmap](#)
- [WinDrawBorder](#)
- [WinDrawPointer](#)
- [WinDrawText](#)
- [WinFillRect](#)
- [WinGetSysBitmap](#)
- [WinInvertRect](#)
- [WinQueryPresParam](#)
- [WinRemovePresParam](#)
- [WinScrollWindow](#)
- [WinSetPresParam](#)

WinScrollWindow - Related Messages

Related Messages

- [WM_PAINT](#)

WinScrollWindow - Example Code

This example shows a very small part of the processing that must be done for a WM_VSCROLL message, which will be sent when a vertical scroll bar has a significant event to notify its owner.

```
#define INCL_WINSCROLLBARS
#define INCL_WINWINDOWMGR
#include <OS2.H>
#define COUNT 10

HWND    hwndClient;
MPARAM  mp2;
ULONG   msg;

switch(msg)
{

case WM_VSCROLL:

    switch (SHORT2FROMMP(mp2))

        case SB_LINEUP:      /* Sent if the operator      */
                                /* clicks on the up arrow    */
                                /* of the scroll bar, or     */
                                /* presses the VK_UP key. */

                                WinScrollWindow(hwndClient,
                                                0,
                                                (LONG)20, /* vertical scroll */
                                                (PRECTL)NULL,
                                                (PRECTL)NULL,
                                                (HRGN)NULLHANDLE,
                                                (PRECTL)NULL,
                                                0);

                                break;

                                .
                                .
                                .

        break;

}
```

WinScrollWindow - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinSendDlgItemMsg

WinSendDlgItemMsg - Syntax

This function sends a message to the dialog item defined by *idItem* in the dialog window specified by *hwndDlg*.

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndDlg; /* Parent-window handle. */
ULONG     idItem;  /* Identity of the child window. */
ULONG     msg;     /* Message identity. */
MPARAM    mp1;     /* Message parameter 1. */
MPARAM    mp2;     /* Message parameter 2. */
MRESULT   mresReply; /* Message-return data. */

mresReply = WinSendDlgItemMsg(hwndDlg, idItem,
                              msg, mp1, mp2);
```

WinSendDlgItemMsg Parameter - hwndDlg

hwndDlg (**HWND**) - input
Parent-window handle.

WinSendDlgItemMsg Parameter - idItem

idItem (**ULONG**) - input
Identity of the child window.

It must be greater or equal to 0 and less or equal to 0xFFFF.

WinSendDlgItemMsg Parameter - msg

msg (**ULONG**) - input
Message identity.

WinSendDlgItemMsg Parameter - mp1

mp1 ([MPARAM](#)) - input
Message parameter 1.

WinSendDlgItemMsg Parameter - mp2

mp2 ([MPARAM](#)) - input
Message parameter 2.

WinSendDlgItemMsg Return Value - mresReply

mresReply ([MRESULT](#)) - returns
Message-return data.

WinSendDlgItemMsg - Parameters

hwndDlg ([HWND](#)) - input
Parent-window handle.

idItem ([ULONG](#)) - input
Identity of the child window.

It must be greater or equal to 0 and less or equal to 0xFFFF.

msg ([ULONG](#)) - input
Message identity.

mp1 ([MPARAM](#)) - input
Message parameter 1.

mp2 ([MPARAM](#)) - input
Message parameter 2.

mresReply ([MRESULT](#)) - returns
Message-return data.

WinSendDlgItemMsg - Remarks

This function is equivalent to the [WinSendMessage](#) function, in which the receiving window procedure is specified by means of the item identity of the child window and parent-window handle.

It does not return until the message has been processed by the dialog item, whose return value is returned in *mresReply*.

The call is equivalent to:

```
WinSendMessage (WinWindowFromID(hwndDlg, idItem), msgid, param1, param2, reply);
```

This function is valid for any window with children; however, it is typically used for dialog items in a dialog window.

WinSendDlgItemMsg - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinSendDlgItemMsg - Related Functions

Related Functions

- [WinBroadcastMsg](#)
- [WinCreateMsgQueue](#)
- [WinDestroyMsgQueue](#)
- [WinDispatchMsg](#)
- [WinGetDlgMsg](#)
- [WinGetMsg](#)
- [WinInSendMessage](#)
- [WinPeekMsg](#)
- [WinPostMsg](#)
- [WinPostQueueMsg](#)
- [WinQueryMsgPos](#)
- [WinQueryMsgTime](#)
- [WinQueryQueueInfo](#)
- [WinQueryQueueStatus](#)
- [WinSendDlgItemMsg](#)
- [WinSendMessage](#)
- [WinSetClassMsgInterest](#)
- [WinSetMsgInterest](#)
- [WinSetMsgMode](#)
- [WinSetSynchroMode](#)
- [WinWaitMsg](#)

WinSendDlgItemMsg - Example Code

This example processes an application-defined message (IDM_SHOW) and sets a check mark next to the selected item.

```
#define INCL_WIN
#define INCL_WINDIALOGS
#include <OS2.H>
```

```

#define IDM_SHOW 902

HWND hwndFrame;
ULONG msg;
MPARAM mp1;

/* Inside client procedure. */

switch(msg)
{
case WM_COMMAND:
/* The user has chosen a menu item. Process the selection */
/* accordingly. */

switch ( SHORT1FROMMP( mp1 ) )
{

case IDM_SHOW:
WinSendDlgItemMsg(hwndFrame, (ULONG) FID_MENU,
(ULONG) MM_SETITEMATTR,
MPFROM2SHORT(IDM_SHOW, TRUE),
MPFROM2SHORT(MIA_CHECKED,MIA_CHECKED));

break;

}
break;
}

```

WinSendDlgItemMsg - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSendMsg

WinSendMsg - Syntax

This function sends a message with identity *ulMsgid* to *hwnd*, passing *mpParam1* and *mpParam2* as the parameters to the window.

```

#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND      hwnd;          /* Window handle. */
ULONG     ulMsgid;       /* Message identity. */
MPARAM    mpParam1;      /* Parameter 1. */
MPARAM    mpParam2;      /* Parameter 2. */
MRESULT   mresReply;     /* Message-return data. */

```

```
mresReply = WinSendMsg(hwnd, ulMsgid, mpParam1,  
                        mpParam2);
```

WinSendMsg Parameter - hwnd

hwnd (**HWND**) - input
Window handle.

WinSendMsg Parameter - ulMsgid

ulMsgid (**ULONG**) - input
Message identity.

WinSendMsg Parameter - mpParam1

mpParam1 (**MPARAM**) - input
Parameter 1.

WinSendMsg Parameter - mpParam2

mpParam2 (**MPARAM**) - input
Parameter 2.

WinSendMsg Return Value - mresReply

mresReply (**MRESULT**) - returns
Message-return data.

WinSendMsg - Parameters

hwnd ([HWND](#)) - input
Window handle.

ulMsgid ([ULONG](#)) - input
Message identity.

mpParam1 ([MPARAM](#)) - input
Parameter 1.

mpParam2 ([MPARAM](#)) - input
Parameter 2.

mresReply ([MRESULT](#)) - returns
Message-return data.

WinSendMessage - Remarks

mresReply is the value returned by the window procedure that is invoked. For standard window classes, the values of *mresReply* are documented with the message definitions.

This function does not complete until the message has been processed by the window procedure whose return value is returned in *mresReply*.

If the window receiving the message belongs to the same thread, the window function is called immediately as a subroutine. If the window is of another thread or process, the operating system switches to the appropriate thread that enters the necessary window procedure recursively. The message is not placed in the queue of the destination thread.

If a message is sent from one process to another and the message contains a pointer, the receiving process may not have read/write access to the memory referenced by the pointer. If the receiving process is expected to update that memory, this must be done using shared memory. For more information about Dynamic Data Exchange (DDE) and shared memory, see "Dynamic Data Exchange" section of the *Presentation Manager Programming Guide - Advanced Topics*.

WinSendMessage - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_WINDOW_NOT_LOCKED (0x1007)
The window specified in WinSendMessage was not locked.

WinSendMessage - Related Functions

Related Functions

- [WinBroadcastMessage](#)
- [WinCreateMessageQueue](#)
- [WinDestroyMessageQueue](#)
- [WinDispatchMessage](#)
- [WinGetDialogMessage](#)
- [WinGetMessage](#)

- [WinInSendMessage](#)
- [WinPeekMsg](#)
- [WinPostMsg](#)
- [WinPostQueueMsg](#)
- [WinQueryMsgPos](#)
- [WinQueryMsgTime](#)
- [WinQueryQueueInfo](#)
- [WinQueryQueueStatus](#)
- [WinSendDlgItemMsg](#)
- [WinSendMessage](#)
- [WinSetClassMsgInterest](#)
- [WinSetMsgInterest](#)
- [WinSetMsgMode](#)
- [WinSetSynchroMode](#)
- [WinWaitMsg](#)

WinSendMessage - Example Code

This example gets the window handle of the system menu and calls WinSendMessage to send a message to disable the Close menu item.

```
#define INCL_WINMENUS
#define INCL_WINMESSAGEGR
#define INCL_WINFRAMEGR
#include <OS2.H>
HWND hwndDlg;
HWND hwndSysMenu;

hwndSysMenu = WinWindowFromID(hwndDlg, FID_SYSMENU);
WinSendMessage(hwndSysMenu, MM_SETITEMATTR,
    MPFROM2SHORT(SC_CLOSE, TRUE),
    MPFROM2SHORT(MIA_DISABLED, MIA_DISABLED));
```

WinSendMessage - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSetAcceleratorTable

WinSetAcceleratorTable - Syntax

This function sets the window-accelerator, or queue-accelerator table.

```
#define INCL_WINACCELERATORS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
HACCEL    haccelAccel; /* Accelerator-table handle. */
HWND     hwndFrame;    /* Frame-window handle. */
BOOL      rc;          /* Success indicator. */

rc = WinSetAccelTable(hab, haccelAccel, hwndFrame);
```

WinSetAccelTable Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinSetAccelTable Parameter - haccelAccel

haccelAccel (**HACCEL**) - input
Accelerator-table handle.

NULLHANDLE
Remove any accelerator table in effect for the window or the queue

Other
Accelerator-table handle.

WinSetAccelTable Parameter - hwndFrame

hwndFrame (**HWND**) - input
Frame-window handle.

NULLHANDLE
Set the queue-accelerator table

Other
Set the window-accelerator table.

WinSetAccelTable Return Value - rc

rc (**BOOL**) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinSetAccelTable - Parameters

hab ([HAB](#)) - input

Anchor-block handle.

haccelAccel ([HACCEL](#)) - input

Accelerator-table handle.

NULLHANDLE

Remove any accelerator table in effect for the window or the queue

Other

Accelerator-table handle.

hwndFrame ([HWND](#)) - input

Frame-window handle.

NULLHANDLE

Set the queue-accelerator table

Other

Set the window-accelerator table.

rc ([BOOL](#)) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinSetAccelTable - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)

An invalid window handle was specified.

PMERR_INVALID_HACCEL (0x101A)

An invalid accelerator-table handle was specified.

WinSetAccelTable - Related Functions

Related Functions

- [WinCopyAccelTable](#)
- [WinCreateAccelTable](#)
- [WinDestroyAccelTable](#)

- [WinLoadAccelTable](#)
- [WinQueryAccelTable](#)
- [WinSetAccelTable](#)
- [WinTranslateAccel](#)

WinSetAccelTable - Example Code

This example uses the WinSetAccelTable call to remove any accelerator table in effect for the window.

```
#define INCL_WIN
#include <OS2.H>
HWND hwndFrame, hwndClient;
HAB hab;
HACCEL haccel;

hwndFrame = WinQueryWindow(hwndClient,
                           QW_PARENT); /* get handle of parent, */
                                       /* which is frame window. */
WinSetAccelTable(hab,
                 (HACCEL)0, /* remove any accelerator table in */
                   /* effect. */
                 hwndFrame);
```

WinSetAccelTable - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSetActiveWindow

WinSetActiveWindow - Syntax

This function makes the frame window the active window.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND    hwndDesktop; /* Desktop-window handle. */
HWND    hwnd;         /* Window handle. */
BOOL    rc;           /* Active-window-set indicator. */
```

```
rc = WinSetActiveWindow(hwndDeskTop, hwnd);
```

WinSetActiveWindow Parameter - hwndDeskTop

- hwndDeskTop (HWND)** - input
Desktop-window handle.
- HWND_DESKTOP The desktop-window handle
 - Other Specified desktop-window handle.

WinSetActiveWindow Parameter - hwnd

- hwnd (HWND)** - input
Window handle.
- hwnd* is either the frame window or its child. If it is a child, the parent frame window will become the active window.

WinSetActiveWindow Return Value - rc

- rc (BOOL)** - returns
Active-window-set indicator.
- TRUE Active window is set
 - FALSE Active window is not set.

WinSetActiveWindow - Parameters

- hwndDeskTop (HWND)** - input
Desktop-window handle.
- HWND_DESKTOP The desktop-window handle
 - Other Specified desktop-window handle.

- hwnd (HWND)** - input
Window handle.

hwnd is either the frame window or its child. If it is a child, the parent frame window will become the active window.

rc ([BOOL](#)) - returns
Active-window-set indicator.

TRUE

Active window is set

FALSE

Active window is not set.

WinSetActiveWindow - Remarks

This function is equivalent to the [WinFocusChange](#) function, in which the *flFocusChange* parameter is set.

WinSetActiveWindow - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinSetActiveWindow - Related Functions

Related Functions

- [WinGetMinPosition](#)
- [WinQueryActiveWindow](#)
- [WinQueryWindowPos](#)
- [WinSaveWindowPos](#)
- [WinSetActiveWindow](#)
- [WinSetMultWindowPos](#)
- [WinSetWindowPos](#)

WinSetActiveWindow - Example Code

This example uses the WinSetActiveWindow call to make the main window the active window.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HWND hwnd;

WinSetActiveWindow(HWND_DESKTOP,hwnd);
```

WinSetActiveWindow - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinSetCapture

WinSetCapture - Syntax

This function captures all pointing device messages.

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndDesktop; /* Desktop-window handle, or HWND_DESKTOP. */
HWND    hwnd;         /* Handle of the window that is to receive all pointing device messages. */
BOOL    rc;           /* Success indicator. */

rc = WinSetCapture(hwndDesktop, hwnd);
```

WinSetCapture Parameter - hwndDesktop

hwndDesktop ([HWND](#)) - input
Desktop-window handle, or HWND_DESKTOP.

WinSetCapture Parameter - hwnd

hwnd ([HWND](#)) - input
Handle of the window that is to receive all pointing device messages.

hwnd can take the special value HWND_THREADCAPTURE to capture the pointing device to the current thread rather than to a particular window. HWND_THREADCAPTURE is unique among window handles.

If *hwnd* is NULLHANDLE, pointing device capture is released.

WinSetCapture Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE

Successful completion.

FALSE

Error occurred. If the pointing device has already been captured by another thread or window, the call fails. This is to prevent applications from removing the capture from other windows or threads.

WinSetCapture - Parameters

hwnDDesktop ([HWND](#)) - input
Desktop-window handle, or HWND_DESKTOP.

hwnD ([HWND](#)) - input
Handle of the window that is to receive all pointing device messages.

hwnD can take the special value HWND_THREADCAPTURE to capture the pointing device to the current thread rather than to a particular window. HWND_THREADCAPTURE is unique among window handles.

If *hwnD* is NULLHANDLE, pointing device capture is released.

rc ([BOOL](#)) - returns
Success indicator.

TRUE

Successful completion.

FALSE

Error occurred. If the pointing device has already been captured by another thread or window, the call fails. This is to prevent applications from removing the capture from other windows or threads.

WinSetCapture - Remarks

This function assigns the pointing device capture to *hwnD*.

With the pointing device capture set to a window, all pointing device input is directed to that window, regardless of whether the pointing device pointer is over that window.

When this function (*hwnDDesktop*, NULLHANDLE) is called to release the pointing device capture, a [WM_MOUSEMOVE](#) message is posted regardless of whether the pointing device pointer has actually moved. This ensures that the window below the pointing device, at that time, is able to change features, such as the shape of the pointing device pointer.

If this function (*hwnDDesktop*, HWND_THREADCAPTURE) is called, the pointing device is captured to the current thread. Pointing device QMSGs processed in this manner have NULLHANDLE window handles, and the pointing device coordinates are relative to the screen.

This function returns an unlocked window handle.

It must only be called while processing pointing device or keyboard input. A message box or dialog box must not be created while the pointing device is captured.

WinSetCapture - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinSetCapture - Related Functions

Related Functions

- [WinQueryCapture](#)
 - WinSetCapture
-

WinSetCapture - Related Messages

Related Messages

- [WM_MOUSEMOVE](#)
-

WinSetCapture - Example Code

This example uses the WinSetCapture call to capture the mouse until the button is released. The user has selected a specific object with mouse button 2.

```
#define INCL_WININPUT
#include <OS2.H>
HWND hwnd;
USHORT msg;
WinSetCapture(hwnd,HWND_DESKTOP);
switch (msg) {
    case WM_BUTTON2DOWN:

        /******
        /* An object has been picked. Set the mouse capture until
        /* a 'button up' message is detected.
        /******
        if (hwnd != WinQueryFocus(HWND_DESKTOP)){
            WinSetFocus(HWND_DESKTOP, hwnd);
        }
        WinSetCapture(HWND_DESKTOP, hwnd);
        break;
    }
}
```

WinSetCapture - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Related Messages](#)

[Glossary](#)

WinSetClassMsgInterest

WinSetClassMsgInterest - Syntax

This function sets the message interest of a window class.

```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;           /* Anchor-block handle. */
PSZ      pszClassName;  /* Window-class name. */
ULONG    ulMsgClass;    /* Message class to have interest level set. */
LONG     lControl;      /* Interest identifier for the message class. */
BOOL     rc;            /* Interest-changed indicator. */

rc = WinSetClassMsgInterest(hab, pszClassName,
                           ulMsgClass, lControl);
```

WinSetClassMsgInterest Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinSetClassMsgInterest Parameter - pszClassName

pszClassName ([PSZ](#)) - input
Window-class name.

WinSetClassMsgInterest Parameter - ulMsgClass

ulMsgClass ([ULONG](#)) - input
Message class to have interest level set.

msgid A single message identity (for example, [WM_SHOW](#)).

SMIM_ALL All messages (except for WM_QUIT if *lControl* is SMI_AUTODISPATCH or SMI_NOINTEREST).

WinSetClassMsgInterest Parameter - lControl

lControl ([LONG](#)) - input
Interest identifier for the message class.

SMI_INTEREST Interested in the message, or messages

SMI_NOINTEREST Not interested in the message, or messages

SMI_AUTODISPATCH Interested in the message or messages, but they are to be automatically dispatched to the window procedure.

WinSetClassMsgInterest Return Value - rc

rc ([BOOL](#)) - returns
Interest-changed indicator.

TRUE Interest successfully changed

FALSE Interest not successfully changed.

WinSetClassMsgInterest - Parameters

hAb ([HAB](#)) - input
Anchor-block handle.

pszClassName ([PSZ](#)) - input
Window-class name.

ulMsgClass ([ULONG](#)) - input
Message class to have interest level set.

msgid A single message identity (for example, [WM_SHOW](#)).

SMIM_ALL All messages (except for WM_QUIT if *lControl* is SMI_AUTODISPATCH or SMI_NOINTEREST).

IControl ([LONG](#)) - input

Interest identifier for the message class.

SMI_INTEREST

Interested in the message, or messages

SMI_NOINTEREST

Not interested in the message, or messages

SMI_AUTODISPATCH

Interested in the message or messages, but they are to be automatically dispatched to the window procedure.

rc ([BOOL](#)) - returns

Interest-changed indicator.

TRUE

Interest successfully changed

FALSE

Interest not successfully changed.

WinSetClassMsgInterest - Remarks

This function has no effect unless the [MsgControlHook](#) hook, which is invoked by this function, has been set. The interest for [WM_QUIT](#) cannot be set to SMI_AUTODISPATCH using SMIM_ALL, because [WM_QUIT](#) is the normal means of terminating an application. It can be set specifically, if required.

WinSetClassMsgInterest - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)

An invalid window handle was specified.

WinSetClassMsgInterest - Related Functions

Related Functions

- [WinBroadcastMsg](#)
- [WinCreateMsgQueue](#)
- [WinDestroyMsgQueue](#)
- [WinDispatchMsg](#)
- [WinGetDlgMsg](#)
- [WinGetMsg](#)
- [WinInSendMessage](#)
- [WinPeekMsg](#)
- [WinPostMsg](#)
- [WinPostQueueMsg](#)
- [WinQueryMsgPos](#)
- [WinQueryMsgTime](#)
- [WinQueryQueueInfo](#)
- [WinQueryQueueStatus](#)
- [WinSendDlgItemMsg](#)
- [WinSendMessage](#)
- [WinSetClassMsgInterest](#)
- [WinSetMsgInterest](#)
- [WinSetMsgMode](#)

- [WinSetSynchroMode](#)
- [WinWaitMsg](#)

WinSetClassMsgInterest - Example Code

This example uses the WinSetClassMsgInterest call to set the message interest of window class WC_MENU. It allows one to process the messages of this window class in the MsgControlHook procedure.

```
#define INCL_WINMESSAGEGR
#define INCL_WINHOOKS
#define INCL_WINMENUS /* for WC_MENU parameter definition. */
#include <OS2.H>
main()
{
    /* Hook Procedure Prototype */

    BOOL MsgControlHook(HAB hab, LONG idContext, /* this hook can */
                        HWND hwnd, PSZ pszClassname, /* be given any */
                        ULONG ulMsgclass, /* name. */
                        LONG idControl, PBOOL fSuccess);

    HWND hwnd;
    HAB hab;
    BOOL fSuccess;

    /* This function passes the hook procedure address to the system. */

    WinSetHook(hab,
                (HMQ)0,
                MCHK_CLASSMSGINTEREST,
                (PFN)MsgControlHook,
                (HMODULE)0); /* hook is into application queue. */

    /*
    This function sets the message interest of a window class.
    */
    WinSetClassMsgInterest(hab,
                           WC_MENU, /* menu window class. */
                           SMIM_ALL, /* set interest level for all */
                           /* messages. */
                           SMI_AUTODISPATCH); /* interested in the */
                                                /* messages, but they are to */
                                                /* be automatically dispatched */
                                                /* to the window procedure. */

}
/*
This hook allows the call which determine the flow of messages to be
intercepted. It must be present for the WinSetClassMsgInterest
call to have an effect.
*/

BOOL MsgControlHook(HAB hab, LONG idContext, /* this hook can */
                    HWND hwnd, PSZ pszClassname, /* be given any */
                    ULONG ulMsgclass, /* name. */
                    LONG idControl, PBOOL fSuccess)
{
    /* ... */
}
```

WinSetClassMsgInterest - Topics

Select an item:

[Syntax](#)

[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSetClassThunkProc

WinSetClassThunkProc - Syntax

This function associates a pointer-conversion procedure with a window class.

```
#define INCL_WINTHUNKAPI /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

PSZ      pszClassName; /* Window-class name. */
PFN      pfnThunkProc; /* Pointer-conversion procedure identifier. */
BOOL     rc;           /* Success indicator. */

rc = WinSetClassThunkProc(pszClassName, pfnThunkProc);
```

WinSetClassThunkProc Parameter - pszClassName

pszClassName ([PSZ](#)) - input
Window-class name.

WinSetClassThunkProc Parameter - pfnThunkProc

pfnThunkProc ([PFN](#)) - input
Pointer-conversion procedure identifier.

- | | |
|-------|--|
| NULL | Any existing pointer-conversion procedure is dissociated from this class.

By default, a class has no pointer-conversion procedure associated with it. |
| Other | The pointer-conversion procedure to be associated with this class. |
-

WinSetClassThunkProc Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	An error occurred.

WinSetClassThunkProc - Parameters

pszClassName ([PSZ](#)) - input
Window-class name.

pfnThunkProc ([PFN](#)) - input
Pointer-conversion procedure identifier.

NULL	Any existing pointer-conversion procedure is dissociated from this class. By default, a class has no pointer-conversion procedure associated with it.
Other	The pointer-conversion procedure to be associated with this class.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	An error occurred.

WinSetClassThunkProc - Remarks

This function does not alter the pointer-conversion procedure associated with any existing window. It changes the pointer-conversion procedure that will be associated with a window created with a subsequent [WinCreateWindow](#) or [WinCreateStdWindow](#) function.

WinSetClassThunkProc - Related Functions

Related Functions

- [WinQueryClassThunkProc](#)
- [WinQueryWindowModel](#)
- [WinQueryWindowThunkProc](#)
- [WinSetClassThunkProc](#)
- [WinSetWindowThunkProc](#)

WinSetClassThunkProc - Example Code

This example sets the pointer conversion procedure of the window class, given that we have an anchor-block handle.

```
#define INCL_WINWINDOWMGR
#define INCL_WINTHUNKAPI
#include <OS2.H>

LONG thunkpr(LONG *p); /* prototype definition. */
main()
{
    HAB hab;
    PFN pfn;
    char *classname;

    WinQueryClassName(hab,
                      sizeof(classname),
                      classname);

    WinSetClassThunkProc(classname,
                         (PFN)thunkpr);
}

LONG thunkpr(LONG *p)
{
    /* 16-bit to 32-bit pointer conversion procedure. */
}
```

WinSetClassThunkProc - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSetClipbrdData

WinSetClipbrdData - Syntax

This call puts data into the clipboard.

```
#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
```

```

HAB      hab;          /* Anchor-block handle. */
ULONG    ulh;          /* General handle to the data object being set into the clipboard. */
ULONG    ulfmt;        /* Clipboard format of the data object referenced by ulh. */
ULONG    flFmtInfo;    /* Information. */
BOOL      rc;          /* Data-placed indicator. */

rc = WinSetClipbrdData(hab, ulh, ulfmt, flFmtInfo);

```

WinSetClipbrdData Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinSetClipbrdData Parameter - ulh

ulh ([ULONG](#)) - input
General handle to the data object being set into the clipboard.

If NULLHANDLE, a [WM_RENDERFMT](#) message is sent to the clipboard-owner window to render the format when [WinQueryClipbrdData](#) is called with the specified format.

Once the data has been set into the clipboard, this handle can no longer be used by the application.

If CFI_POINTER is specified, this parameter contains a pointer to memory. The memory must have been allocated as unnamed and shareable, by DosAllocSharedMem with the OBJ_GIVEABLE attribute.

WinSetClipbrdData Parameter - ulfmt

ulfmt ([ULONG](#)) - input
Clipboard format of the data object referenced by *ulh*.

The standard clipboard formats are shown in the following list. In addition to these predefined formats, any format value registered through the standard system atom manager displays this format in preference to privately-formatted data.

CF_TEXT	Text format. Each line ends with a carriage-return/line-feed combination. Tab characters separate fields within a line. A NULL character signals the end of the data.
CF_DSPTEXT	Text display format associated with private format.
CF_BITMAP	Bit map.
CF_DSPBITMAP	Bit-map display format associated with private format.
CF_METAFILE	Metafile.
CF_DSPMETAFILE	Metafile display format associated with private format.
CF_PALETTE	Palette.

WinSetClipbrdData Parameter - flFmtInfo

flFmtInfo ([ULONG](#)) - input
Information.

Information about the type of data referenced by the *ul/h* parameter.

Memory Model

One and only one of CFI_POINTER and CFI_HANDLE must be specified, unless CFI_OWNERDISPLAY is also specified.

CFI_POINTER

The *ul/h* parameter is a flat pointer to the object.

When this memory model is specified, the system:

- saves the address (accessing it from the shell process), so that if the setting application terminates normally or abnormally, the data is still available.
- frees the memory from the setting process, so that the setting application may no longer use it.

CFI_POINTER must be specified if the *ul/h* parameter is CF_TEXT or CF_DSPTEXT.

CFI_HANDLE

The *ul/h* parameter is the handle to a metafile or bit map.

This must be specified if the *ul/h* parameter is CF_BITMAP, CF_DSPBITMAP, CF_METAFILE or CF_DSPMETAFILE.

Usage Flags

CFI_OWNERFREE

Handle is not freed by [WinEmptyClipbrd](#). The application must free the data if necessary.

CFI_OWNERDISPLAY

This flag indicates that the format is drawn by the clipboard owner in the clipboard viewer window by means of the [WM_PAINTCLIPBOARD](#) message. The *ul/h* parameter should be NULL.

The values may be combined with bit-wise OR. Any number of the usage flags may be specified, but only one of the memory models.

WinSetClipbrdData Return Value - rc

rc ([BOOL](#)) - returns
Data-placed indicator.

Indicates whether data is placed into clipboard by this call:

TRUE

Data placed into clipboard.

FALSE

Data is not placed into clipboard, either an error occurred, or *ul/h* is NULL.

WinSetClipbrdData - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

ulh ([ULONG](#)) - input
General handle to the data object being set into the clipboard.

If NULLHANDLE, a [WM_RENDERFMT](#) message is sent to the clipboard-owner window to render the format when [WinQueryClipbrdData](#) is called with the specified format.

Once the data has been set into the clipboard, this handle can no longer be used by the application.

If CFI_POINTER is specified, this parameter contains a pointer to memory. The memory must have been allocated as unnamed and shareable, by DosAllocSharedMem with the OBJ_GIVEABLE attribute.

ulfmt ([ULONG](#)) - input
Clipboard format of the data object referenced by *ulh*.

The standard clipboard formats are shown in the following list. In addition to these predefined formats, any format value registered through the standard system atom manager displays this format in preference to privately-formatted data.

CF_TEXT	Text format. Each line ends with a carriage-return/line-feed combination. Tab characters separate fields within a line. A NULL character signals the end of the data.
CF_DSPTEXT	Text display format associated with private format.
CF_BITMAP	Bit map.
CF_DSPBITMAP	Bit-map display format associated with private format.
CF_METAFILE	Metafile.
CF_DSPMETAFILE	Metafile display format associated with private format.
CF_PALETTE	Palette.

flFmtInfo ([ULONG](#)) - input
Information.

Information about the type of data referenced by the *ulh* parameter.

Memory Model

One and only one of CFI_POINTER and CFI_HANDLE must be specified, unless CFI_OWNERDISPLAY is also specified.

CFI_POINTER	The <i>ulh</i> parameter is a flat pointer to the object. When this memory model is specified, the system: <ul style="list-style-type: none">saves the address (accessing it from the shell process), so that if the setting application terminates normally or abnormally, the data is still available.frees the memory from the setting process, so that the setting application may no longer use it.
-------------	--

CFI_POINTER must be specified if the *ulfmt* parameter is CF_TEXT or CF_DSPTEXT.

CFI_HANDLE	The <i>ulh</i> parameter is the handle to a metafile or bit map.
------------	--

This must be specified if the *ulfmt* parameter is CF_BITMAP, CF_DSPBITMAP, CF_METAFILE or CF_DSPMETAFILE.

Usage Flags

CFI_OWNERFREE	Handle is not freed by WinEmptyClipbrd . The application must free the data if necessary.
CFI_OWNERDISPLAY	This flag indicates that the format is drawn by the clipboard owner in the clipboard viewer window by means of the WM_PAINTCLIPBOARD message. The <i>ulh</i> parameter should be NULL.

The values may be combined with bit-wise OR. Any number of the usage flags may be specified, but only one of the memory models.

rc ([BOOL](#)) - returns
Data-placed indicator.

Indicates whether data is placed into clipboard by this call:

TRUE	Data placed into clipboard.
FALSE	Data is not placed into clipboard, either an error occurred, or <i>///</i> is NULL.

WinSetClipbrdData - Remarks

Data of the specified format, already in the clipboard, is freed by this call.

An object passed to the clipboard becomes the property of the system, and is not deleted when the process that created it terminates.

WinSetClipbrdData - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_FLAG (0x1019)
An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

PMERR_INVALID_INTEGER_ATOM (0x1016)
The specified atom is not a valid integer atom.

WinSetClipbrdData - Related Functions

Related Functions

- [WinCloseClipbrd](#)
- [WinEmptyClipbrd](#)
- [WinEnumClipbrdFmts](#)
- [WinOpenClipbrd](#)
- [WinQueryClipbrdData](#)
- [WinQueryClipbrdFmtInfo](#)
- [WinQueryClipbrdOwner](#)
- [WinQueryClipbrdViewer](#)
- [WinSetClipbrdData](#)
- [WinSetClipbrdOwner](#)
- [WinSetClipbrdViewer](#)

WinSetClipbrdData - Related Messages

Related Messages

- [WM_RENDERFMT](#)
- [WM_PAINTCLIPBOARD](#)

WinSetClipbrdData - Example Code

This example puts a bit map into the clipboard.

```
#define INCL_WINCLIPBOARD
#include <OS2.H>
HAB hab;          /* anchor-block handle. */
HBITMAP bmap;     /* bit-map handle.      */

WinOpenClipbrd(hab);
WinSetClipbrdData(hab,
                  (ULONG)bmap,
                  CF_BITMAP,
                  CFI_HANDLE); /* tells the system that the */
                               /* bmap parameter is a handle */
                               /* to a bit map.          */
WinCloseClipbrd(hab);
```

WinSetClipbrdData - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinSetClipbrdOwner

WinSetClipbrdOwner - Syntax

This function sets the current clipboard-owner window.

```
#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB    hab;  /* Anchor-block handle. */
HWND   hwnd; /* Window handle of the new clipboard owner. */
BOOL   rc;   /* Success indicator. */
```

```
rc = WinSetClipbrdOwner(hab, hwnd);
```

WinSetClipbrdOwner Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinSetClipbrdOwner Parameter - hwnd

hwnd (**HWND**) - input
Window handle of the new clipboard owner.

NULLHANDLE	Clipboard-owner window is released and no new clipboard-owner window is established.
Other	Window handle of the new clipboard owner.

WinSetClipbrdOwner Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetClipbrdOwner - Parameters

hab (**HAB**) - input
Anchor-block handle.

hwnd (**HWND**) - input
Window handle of the new clipboard owner.

NULLHANDLE	Clipboard-owner window is released and no new clipboard-owner window is established.
Other	Window handle of the new clipboard owner.

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetClipbrdOwner - Remarks

The clipboard owner window receives the following clipboard-related messages at appropriate times:

[WM_DESTROYCLIPBOARD](#)
[WM_HSCROLLCLIPBOARD](#)
[WM_PAINTCLIPBOARD](#)
[WM_RENDERALLFMTS](#)
[WM_RENDERFMT](#)
[WM_SIZECLIPBOARD](#)
[WM_VSCROLLCLIPBOARD](#).

WinSetClipbrdOwner - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
 An invalid window handle was specified.

WinSetClipbrdOwner - Related Functions

Related Functions

- [WinCloseClipbrd](#)
- [WinEmptyClipbrd](#)
- [WinEnumClipbrdFmts](#)
- [WinOpenClipbrd](#)
- [WinQueryClipbrdData](#)
- [WinQueryClipbrdFmtInfo](#)
- [WinQueryClipbrdOwner](#)
- [WinQueryClipbrdViewer](#)
- [WinSetClipbrdData](#)
- [WinSetClipbrdOwner](#)
- [WinSetClipbrdViewer](#)

WinSetClipbrdOwner - Related Messages

Related Messages

- [WM_DESTROYCLIPBOARD](#)
- [WM_HSCROLLCLIPBOARD](#)
- [WM_PAINTCLIPBOARD](#)

- [WM_RENDERALLFMTS](#)
- [WM_RENDERFMT](#)
- [WM_SIZECLIPBOARD](#)
- [WM_VSCROLLCLIPBOARD](#)

WinSetClipbrdOwner - Example Code

This example places a bit map into the clipboard.

```
#define INCL_WINCLIPBOARD
#include <OS2.H>
HAB hab;           /* anchor-block handle. */
HBITMAP bmap;      /* bit-map handle. */
HWND hwnd;

WinOpenClipbrd(hab);
WinSetClipbrdOwner(hab,
                   hwnd); /* window handle of the clipboard */
                          /* owner. */
WinSetClipbrdData(hab,
                  (ULONG)bmap,
                  CF_BITMAP,
                  CFI_HANDLE); /* tells the system that the */
                               /* bmap parameter is a handle */
                               /* to a bit map. */
WinCloseClipbrd(hab);
```

WinSetClipbrdOwner - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinSetClipbrdViewer

WinSetClipbrdViewer - Syntax

This function sets the current clipboard-viewer window to a specified window.

```

#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
HWND     hwndNewClipViewer; /* Window handle of the new clipboard viewer. */
BOOL     rc;           /* Success indicator. */

rc = WinSetClipbrdViewer(hab, hwndNewClipViewer);

```

WinSetClipbrdViewer Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinSetClipbrdViewer Parameter - hwndNewClipViewer

hwndNewClipViewer (**HWND**) - input
Window handle of the new clipboard viewer.

NULLHANDLE	The clipboard-viewer window is released and no new clipboard-viewer window is established.
Other	Window handle of the new clipboard viewer.

WinSetClipbrdViewer Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Valid, new clipboard-viewer window established
FALSE	There is no new clipboard-viewer window established.

WinSetClipbrdViewer - Parameters

hab (**HAB**) - input
Anchor-block handle.

hwndNewClipViewer (**HWND**) - input
Window handle of the new clipboard viewer.

NULLHANDLE

The clipboard-viewer window is released and no new clipboard-viewer window is established.

Other

Window handle of the new clipboard viewer.

rc ([BOOL](#)) - returns
Success indicator.

TRUE

Valid, new clipboard-viewer window established

FALSE

There is no new clipboard-viewer window established.

WinSetClipbrdViewer - Remarks

The clipboard-viewer window receives the [WM_DRAWCLIPBOARD](#) message when the contents of the clipboard change. This allows the viewer window to display an up-to-date version of the clipboard contents.

The clipboard must be open before this function is invoked.

WinSetClipbrdViewer - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinSetClipbrdViewer - Related Functions

Related Functions

- [WinCloseClipbrd](#)
- [WinEmptyClipbrd](#)
- [WinEnumClipbrdFmts](#)
- [WinOpenClipbrd](#)
- [WinQueryClipbrdData](#)
- [WinQueryClipbrdFmtInfo](#)
- [WinQueryClipbrdOwner](#)
- [WinQueryClipbrdViewer](#)
- [WinSetClipbrdData](#)
- [WinSetClipbrdOwner](#)
- [WinSetClipbrdViewer](#)

WinSetClipbrdViewer - Related Messages

Related Messages

- [WM_DRAWCLIPBOARD](#)

WinSetClipbrdViewer - Example Code

This example shows how a window views the clipboard contents.

```
#define INCL_WINCLIPBOARD
#include <OS2.H>
ULONG hclipbrdData;
HAB hab; /* anchor-block handle. */
HBITMAP bmap; /* bit-map handle. */
HWND hwnd;

WinOpenClipbrd(hab);
WinSetClipbrdViewer(hab,
                    hwnd); /* window handle of the clipboard */
                        /* viewer. */
hclipbrdData = WinQueryClipbrdData(hab,
                                    CF_TEXT);
WinCloseClipbrd(hab);
```

WinSetClipbrdViewer - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Errors](#)
- [Remarks](#)
- [Example Code](#)
- [Related Functions](#)
- [Related Messages](#)
- [Glossary](#)

WinSetControlColors

WinSetControlColors - Syntax

This function sets the colors to be used by a control window. See [Colors Used by PM Controls](#).

```
#define INCL_WINSYS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwnd; /* Window handle. */
LONG      clrType; /* Control color index. */
ULONG     fCtlColor; /* Control Color Flags. */
ULONG     cCtlColor; /* Number of items being passed in pCtlColor. */
```



```
PCTLCOLOR    pCtlColor; /* Array of control colors, each comprising an index and value pair. */
LONG         rc;         /* Number of control colors returned in pCtlColor. */

rc = WinSetControlColors(hwnd, clrType, fCtlColor,
                        cCtlColor, pCtlColor);
```

WinSetControlColors Parameter - hwnd

hwnd ([HWND](#)) - input
Window handle.

WinSetControlColors Parameter - clrType

clrType ([LONG](#)) - input
Control color index.

Must be one of the following control color indexes.

CCT_STATIC	Static bitmap.
CCT_STATICTEXT	Static text.
CCT_GROUPBOX	Group Box.
CCT_PUSHBUTTON	Push button.
CCT_CHECKBOX	Check box.
CCT_RADIOBUTTON	Radio button.
CCT_ENTRYFIELD	Entry field.
CCT_LISTBOX	List box.
CCT_COMBOBOX	Combo box.
CCT_SCROLLBAR	Scroll bar.
CCT_FRAME	Window frame.
CCT_MENU	Menu.
CCT_TITLEBAR	Title bar.
CCT_SPINBUTTON	Spin button.
CCT_SLIDER	Slider.
CCT_CIRCULARSLIDER	Circular slider.
CCT_VALUESET	Value set.
CCT_MLE	Multiple line entry field.
CCT_CONTAINER	Container.
CCT_NOTEBOOK	Notebook.

WinSetControlColors Parameter - fCtlColor

fCtlColor (**ULONG**) - input
Control Color Flags.

CCF_GLOBAL Set global default colors (default)

CCF_APPLICATION Set application default colors (used on current thread)

WinSetControlColors Parameter - cCtlColor

cCtlColor (**ULONG**) - input
Number of items being passed in *pCtlColor*.

WinSetControlColors Parameter - pCtlColor

pCtlColor (**PCTLCOLOR**) - in/out
Array of control colors, each comprising an index and value pair.

WinSetControlColors Return Value - rc

rc (**LONG**) - returns
Number of control colors returned in *pCtlColor*.

TRUE Successful completion.

FALSE Error occurred, invalid parameters.

WinSetControlColors - Parameters

hwnd (**HWND**) - input
Window handle.

clrType (**LONG**) - input
Control color index.

Must be one of the following control color indexes.

CCT_STATIC	Static bitmap.
CCT_STATICTEXT	Static text.
CCT_GROUPBOX	Group Box.
CCT_PUSHBUTTON	Push button.
CCT_CHECKBOX	Check box.
CCT_RADIOBUTTON	Radio button.
CCT_ENTRYFIELD	Entry field.
CCT_LISTBOX	List box.
CCT_COMBOBOX	Combo box.
CCT_SCROLLBAR	Scroll bar.
CCT_FRAME	Window frame.
CCT_MENU	Menu.
CCT_TITLEBAR	Title bar.
CCT_SPINBUTTON	Spin button.
CCT_SLIDER	Slider.
CCT_CIRCULARSLIDER	Circular slider.
CCT_VALUESET	Value set.
CCT_MLE	Multiple line entry field.
CCT_CONTAINER	Container.
CCT_NOTEBOOK	Notebook.

fCtlColor (**ULONG**) - input
Control Color Flags.

CCF_GLOBAL	Set global default colors (default)
CCF_APPLICATION	Set application default colors (used on current thread)

cCtlColor (**ULONG**) - input
Number of items being passed in *pCtlColor*.

pCtlColor (**PCTLCOLOR**) - in/out
Array of control colors, each comprising an index and value pair.

rc (**LONG**) - returns
Number of control colors returned in *pCtlColor*.

TRUE	Successful completion.
FALSE	Error occurred, invalid parameters.

WinSetControlColors - Remarks

This function sets one or more of the colors to be used by a control window of the type specified by *ColorType*. If *hwnd* is a valid control window handle, the colors are set as presentation parameters for that window. If *hwnd* is set to `HWND_DESKTOP`, the default colors being used by that type of control window are set.

To set the default control colors for the current thread, specify `CCF_APPLICATION`. Be sure to issue the call to `WinSetColors` from the same thread that created the control windows, otherwise the call to `WinSetControlColors` will have no effect. To set all the colors used by a control window at once, specify `CCF_GLOBAL`, which is the default.

Each item in the array should contain a valid index and value pair (`CTLCOLOR`). The control color index must be a valid `CCI_` constant. The control color value can be one of the following:

- A valid RGB color

Note: Valid RGB colors include `SYSCLR_` values, which are actually indexes but can be used by graphics functions in RGB mode.
- `CCV_DEFAULT`, which sets the color to its default value
- `CCV_IGNORE`, which leaves the color for this index unchanged.

If `CCV_DEFAULT` is used when *hwnd* is a valid control window, the presentation parameter that maps to the specified `CCI_` index is removed.

If an RGB value that is not a pure color is specified as one of the color values, and if that color is used to draw lines with the control, the end result is not defined. Although dithered colors are okay for filled areas such as backgrounds, they cannot be used for lines.

This function can only be used for setting the colors of standard PM controls.

WinSetControlColors - Related Functions

Related Functions

- [WinQueryControlColors](#)

WinSetControlColors - Example Code

This example sets new border colors for all the push buttons in the application.

```

/*****
/* Query the number of colors used by a push button.
*****/

cCount = WinQueryControlColors(
    HWND_DESKTOP,          /* Desktop window handle */
    CCT_PUSHBUTTON,        /* Select push button */
    CCF_COUNTCOLORS,       /* Count number of colors */
    0, 0);

pactlColor = (PCTLCOLOR)malloc(sizeof(CTLCOLOR) * cCount);

/*****
/* Query all the colors used by push buttons in application.
*****/

WinQueryControlColors(HWND_DESKTOP,          /* Desktop window handle */
    CCT_PUSHBUTTON,        /* Select push button */
    CCF_ALLCOLORS |        /* Return all colors ... */
    CCF_APPLICATION,       /* ... for application */
    cCount,                /* Size of array */
    pactlColor);           /* Buffer for color data */
```

```

/*****
/* Give the global push button borders a red color. */
*****/

for (i = 0; i < cCount; i++)
{
    switch (pactlColor[i].clrIndex)
    {
        case CCI_BORDERLIGHT:

            pactlColor[i].clrValue = 0x00FFC0C0; /* Light red */
            break;

        case CCI_BORDERDARK:

            pactlColor[i].clrValue = 0x00C00000; /* Dark red */
            break;

        default:

            pactlColor[i].clrValue = CCV_DEFAULT; /* Default color */
            break;
    }
}
/*****
/* Set the new border colors for all push buttons in application. */
*****/

WinSetControlColors(HWND_DESKTOP, /* Desktop window handle */
    CCT_PUSHBUTTON, /* Select push button */
    CCF_APPLICATION, /* Application colors */
    cCount, /* Number of colors */
    pactlColor); /* Buffer for color data */

```

WinSetControlColors - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSetCp

WinSetCp - Syntax

This function sets the code page for a queue.

```

#define INCL_WINCOUNTRY /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

```

```
HMQ      hmq;          /* Message-queue handle. */
ULONG    ulCodePage;   /* Code page. */
BOOL      rc;          /* Success indicator. */

rc = WinSetCp(hmq, ulCodePage);
```

WinSetCp Parameter - hmq

hmq (**HMQ**) - input
Message-queue handle.

WinSetCp Parameter - ulCodePage

ulCodePage (**ULONG**) - input
Code page.

Either of the two ASCII code pages specified in CONFIG.SYS can be selected.

WinSetCp Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetCp - Parameters

hmq (**HMQ**) - input
Message-queue handle.

ulCodePage (**ULONG**) - input
Code page.

Either of the two ASCII code pages specified in CONFIG.SYS can be selected.

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
------	-----------------------

FALSE

Error occurred.

WinSetCp - Remarks

The following is the list of valid code pages:

Country	Code Page
Canadian-French	863
Desktop Publishing	1004
Iceland	861
Latin 1 Multilingual	850
Latin 2 Multilingual	852
Nordic	865
Portuguese	860
Turkey	857
U.S. (IBM PC)	437

Code page 1004 is compatible with Microsoft** Windows**.

The following EBCDIC code pages, based on character set 697, are also available for output:

Country	Code Page
Austrian/German	273
Belgian	500
Brazil	037
Czechoslovakia	870
Danish/Norwegian	277
Finnish/Swedish	278
French	297
Hungary	870
Iceland	871
International	500
Italian	280
Poland	870
Portuguese	037
Spanish	284
Turkey	1026
U.K.-English	285
U.S.-English	037
Yugoslavia	870

Code pages 274 (Belgian) and 282 (Portuguese) can be used to provide access to old data.

WinSetCp - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HMQ (0x1002)

An invalid message-queue handle was specified.

PMERR_RESOURCE_NOT_FOUND (0x100A)

The specified resource identity could not be found.

WinSetCp - Related Functions

Related Functions

- [WinCpTranslateChar](#)
- [WinCpTranslateString](#)
- [WinQueryCp](#)
- [WinQueryCpList](#)
- [WinSetCp](#)

WinSetCp - Example Code

This example sets the code page for a message queue to 850 if it is not already set.

```
#define INCL_WINCOUNTRY
#include <OS2.H>
HMQ hmq;

if(WinQueryCp(hmq) != 850)
{
    WinSetCp(hmq, 850);
}
```

WinSetCp - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSetDesktopBkgnd

WinSetDesktopBkgnd - Syntax

This function sets the desktop window state.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndDesktop; /* Desktop-window handle. */
PDESKTOP  pDesktopState; /* Desktop-state structure. */
HBITMAP    hbm; /* Desktop background bit-map handle loaded or set. */
```



```
hbm = WinSetDesktopBkgnd(hwndDesktop, pDesktopState);
```

WinSetDesktopBkgnd Parameter - hwndDesktop

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

WinSetDesktopBkgnd Parameter - pDesktopState

pDesktopState ([PDESKTOP](#)) - input
Desktop-state structure.

If the flag is set, then the background is unaffected although the bit-map file may still be loaded and tiled, or scaled as requested.

WinSetDesktopBkgnd Return Value - hbm

hbm ([HBITMAP](#)) - returns
Desktop background bit-map handle loaded or set.

NULLHANDLE
Error occurred.

Other
Bit-map handle loaded, set, or both for desktop background. The bit-map handle returned may be different from that passed into call if any scaling or tiling is performed.

WinSetDesktopBkgnd - Parameters

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

pDesktopState ([PDESKTOP](#)) - input
Desktop-state structure.

If the flag is set, then the background is unaffected although the bit-map file may still be loaded and tiled, or scaled as requested.

hbm ([HBITMAP](#)) - returns
Desktop background bit-map handle loaded or set.

NULLHANDLE
Error occurred.

Other

Bit-map handle loaded, set, or both for desktop background. The bit-map handle returned may be different from that passed into call if any scaling or tiling is performed.

WinSetDesktopBkgnd - Remarks

This function allows an application to present an image in the background of the desktop window. This application must be acting as the OS/2 PM shell in place of the IBM supplied shell. If the IBM supplied shell is executing it maintains control of the background of the desktop window, and WinSetDesktopBkgnd will have no effect on the desktop window background, but will indicate a successful return code. The background of the desktop window is that portion of the desktop on which no other windows have been painted.

The system assumes ownership of the bit map which forms the desktop background. This implies that once the bit map is set to form the desktop background, it is no longer available to an application and therefore must not be associated with any application presentation space or any symbol set LCID. The system repaints the desktop background automatically to show any changes.

The most recent invocation of this function sets the state of the desktop background. Consequently, any application which sets the desktop background must be aware that changing the desktop background every time the application is activated, which implies the repainting of the whole desktop, could be distracting, if not disorienting, to the user. Therefore, such an application should determine if the correct desktop background is already showing by processing the [WM_ACTIVATE](#) message and if its determining the desktop background state by using the [WinQueryDesktopBkgnd](#) function and checking the bit-map handle of the current desktop background with the desired bit-map handle.

When setting a new desktop background, it is important to ensure that any previous desktop background bit map is destroyed, in order to prevent the system becoming cluttered with unused bit maps.

WinSetDesktopBkgnd - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HPTR (0x101B)
An invalid pointer handle was specified.

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinSetDesktopBkgnd - Related Functions

Related Functions

- [WinQueryDesktopBkgnd](#)

WinSetDesktopBkgnd - Example Code

This example sets the desktop background with a bit map if it is not already set.

```
#define INCL_WINDESKTOP
#define INCL_WINWINDOWMGR
#include <OS2.H>
```

```
HWND    hwndDesktop;
```

```

HAB      hab;
DESKTOP DeskTopState;
HBITMAP hbm;
HBITMAP hbm_user;

WinQueryDesktopBkgnd(HWND_DESKTOP,
                    &DeskTopState);

if (hbm_user != DeskTopState.hbm)
{
    DeskTopState.fl = SDT_LOADFILE;
    /* the szFile is used to load the bit map because*/
    /* the fl parameter is set to SDT_LOADFILE.      */
    strcpy(DeskTopState.szFile, "fruit.bmp");
    DeskTopState.hbm = hbm_user;
    WinSetDesktopBkgnd(hwndDesktop,
                      &DeskTopState);
}

```

WinSetDesktopBkgnd - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSetDlgItemShort

WinSetDlgItemShort - Syntax

This function converts an integer value into the text of a dialog item.

```

#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND      hwndDlg; /* Parent-window handle. */
ULONG     idItem;  /* Identity of the child window whose text is to be changed. */
USHORT    usValue; /* Integer value used to generate the dialog item text. */
BOOL      fSigned; /* Sign indicator. */
BOOL      rc;      /* Success indicator. */

rc = WinSetDlgItemShort(hwndDlg, idItem, usValue,
                        fSigned);

```

WinSetDlgItemShort Parameter - hwndDlg

hwndDlg (**HWND**) - input
Parent-window handle.

WinSetDlgItemShort Parameter - idItem

idItem (**ULONG**) - input
Identity of the child window whose text is to be changed.

It must be greater or equal to 0 and less or equal to 0xFFFF.

WinSetDlgItemShort Parameter - usValue

usValue (**USHORT**) - input
Integer value used to generate the dialog item text.

WinSetDlgItemShort Parameter - fSigned

fSigned (**BOOL**) - input
Sign indicator.

TRUE	Signed integer value
FALSE	Unsigned integer value.

WinSetDlgItemShort Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetDlgItemShort - Parameters

hwndDlg ([HWND](#)) - input
Parent-window handle.

idItem ([ULONG](#)) - input
Identity of the child window whose text is to be changed.

It must be greater or equal to 0 and less or equal to 0xFFFF.

usValue ([USHORT](#)) - input
Integer value used to generate the dialog item text.

fSigned ([BOOL](#)) - input
Sign indicator.

TRUE Signed integer value
FALSE Unsigned integer value.

rc ([BOOL](#)) - returns
Success indicator.

TRUE Successful completion
FALSE Error occurred.

WinSetDlgItemShort - Remarks

The text produced is an ASCII string.

This function is valid for any window with children; however, it is typically used for dialog items in a dialog window.

WinSetDlgItemShort - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinSetDlgItemShort - Related Functions

Related Functions

- [WinQueryDlgItemShort](#)
- [WinQueryDlgItemText](#)
- [WinQueryDlgItemTextLength](#)

- [WinQueryWindowText](#)
- [WinQueryWindowTextLength](#)
- [WinSetDlgItemShort](#)
- [WinSetDlgItemText](#)
- [WinSetWindowText](#)

WinSetDlgItemShort - Example Code

This example gets the text from a Dialog Box entry field as an integer value.

```
#define INCL_WINDIALOGS
#define INCL_WINBUTTONS
#include <OS2.H>
#define ID_ENTRYFLD 900
#define EM_SETTEXTLIMIT 2
HAB hab;
HWND hwnd;
ULONG msg;

switch(msg)
{
    case WM_INITDLG:

/* set entry field text limit. */
    WinSendDlgItemMsg(hwnd,
        /* identifier of the entry field window, which is */
        /* a child of the the window defined by hwnd.      */
        (ULONG)ID_ENTRYFLD,
        (ULONG)EM_SETTEXTLIMIT, /* Limit length */
        /* MPFROM2SHORT macro is of the form (low 2 bytes, */
        /* high 2 bytes), the the number passed is simply 2. */
        MPFROM2SHORT(2,0),
        (MPARAM)0);

/* set entry field to 12. */
    WinSetDlgItemShort(hwnd, ID_ENTRYFLD, (SHORT)12,TRUE);
}
```

WinSetDlgItemShort - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSetDlgItemText

WinSetDlgItemText - Syntax

This function sets a text string in a dialog item.

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND     hwndDlg; /* Parent-window handle. */
ULONG     idItem; /* Identity of the child window whose text is to be set. */
PSZ       pszText; /* Source string. */
BOOL      rc;      /* Success indicator. */

rc = WinSetDlgItemText(hwndDlg, idItem, pszText);
```

WinSetDlgItemText Parameter - hwndDlg

hwndDlg (**HWND**) - input
Parent-window handle.

WinSetDlgItemText Parameter - idItem

idItem (**ULONG**) - input
Identity of the child window whose text is to be set.

It must be greater or equal to 0 and less or equal to 0xFFFF.

WinSetDlgItemText Parameter - pszText

pszText (**PSZ**) - input
Source string.

This is the text string that is to be set into the dialog item.

WinSetDlgItemText Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetDlgItemText - Parameters

hwndDlg ([HWND](#)) - input
Parent-window handle.

idItem ([ULONG](#)) - input
Identity of the child window whose text is to be set.

It must be greater or equal to 0 and less or equal to 0xFFFF.

pszText ([PSZ](#)) - input
Source string.

This is the text string that is to be set into the dialog item.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetDlgItemText - Remarks

This function is valid for any window with children. However, it is typically used for dialog items in a dialog window.

This function is equivalent to:

```
WinSetWindowText (WinWindowFromID (hwndDlg, idItem), pszText);
```

WinSetDlgItemText - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinSetDlgItemText - Related Functions

Related Functions

- [WinQueryDlgItemShort](#)
- [WinQueryDlgItemText](#)
- [WinQueryDlgItemTextLength](#)
- [WinQueryWindowText](#)
- [WinQueryWindowTextLength](#)
- [WinSetDlgItemShort](#)
- [WinSetDlgItemText](#)
- [WinSetWindowText](#)

WinSetDlgItemText - Example Code

This example sets the text "CALENDAR" in a dialog box.

```
#define INCL_WINDIALOGS
#include <OS2.H>
#define ID_DLG_CALENDAR 900
HWND    hwndDlg;

WinSetDlgItemText(hwndDlg,
                  ID_DLG_CALENDAR,
                  "CALENDAR" );
```

WinSetDlgItemText - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSetFileIcon

WinSetFileIcon - Syntax

WinSetFileIcon sets the icon for the file specified by *pFileName* to the icon specified by *picon*.

```
#define INCL_WINPOINTERS
#include <os2.h>

PSZ      pFileName; /* Pointer to file name. */
PICONINFO picon;    /* A pointer to an ICONINFO structure containing an icon specification. */
```

```
BOOL rc; /* Success indicator. */  
rc = WinSetFileIcon(pFileName, picon);
```

WinSetFileIcon Parameter - pFileName

pFileName ([PSZ](#)) - input
Pointer to file name.

A pointer to a zero-terminated string which contains the name of the file whose icon will be set.

WinSetFileIcon Parameter - picon

picon ([PICONINFO](#)) - input
A pointer to an [ICONINFO](#) structure containing an icon specification.

WinSetFileIcon Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetFileIcon - Parameters

pFileName ([PSZ](#)) - input
Pointer to file name.

A pointer to a zero-terminated string which contains the name of the file whose icon will be set.

picon ([PICONINFO](#)) - input
A pointer to an [ICONINFO](#) structure containing an icon specification.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetFileIcon - Remarks

The specified icon is written to the file's .ICON extended attribute.

WinSetFileIcon - Related Functions

Related Functions

- [WinLoadFileIcon](#)
 - [WinFreeFileIcon](#)
-

WinSetFileIcon - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

WinSetFocus

WinSetFocus - Syntax

This function sets the focus window.

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND     hwndDeskTop; /* Desktop-window handle. */
HWND     hwndNewFocus; /* Window handle to receive the focus. */
BOOL     rc;           /* Success indicator. */

rc = WinSetFocus(hwndDeskTop, hwndNewFocus);
```

WinSetFocus Parameter - hwndDesktop

hwndDesktop (HWND) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

WinSetFocus Parameter - hwndNewFocus

hwndNewFocus (HWND) - input
Window handle to receive the focus.

If *hwndNewFocus* identifies a desktop window, no window on the device associated with the *hwndDesktop* receives the focus.

WinSetFocus Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetFocus - Parameters

hwndDesktop (HWND) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

hwndNewFocus (HWND) - input
Window handle to receive the focus.

If *hwndNewFocus* identifies a desktop window, no window on the device associated with the *hwndDesktop* receives the focus.

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
------	-----------------------

FALSE

Error occurred.

WinSetFocus - Remarks

This function is equivalent to the [WinFocusChange](#) call in which the *flFocusChange* parameter is set to 0.

If no window has the input focus, [WM_CHAR](#) messages are posted to the queue of the active window and are not thrown away.

When this function is called a [WM_MOUSEMOVE](#) message is posted regardless of whether the pointing device pointer has actually moved. This ensures that the window below the pointing device, at that time, is able to change features, such as the shape of the pointing device pointer.

This function requires the existence of a message queue.

WinSetFocus - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_CANNOT_SET_FOCUS (0x1037)
Focus cannot be set if a focus change is in progress, or if a system-modal window exists.

WinSetFocus - Related Functions

Related Functions

- [WinEnablePhysInput](#)
- [WinFocusChange](#)
- [WinGetKeyState](#)
- [WinGetPhysKeyState](#)
- [WinQueryFocus](#)
- [WinSetFocus](#)
- [WinSetKeyboardStateTable](#)

WinSetFocus - Related Messages

Related Messages

- [WM_CHAR](#)
- [WM_MOUSEMOVE](#)

WinSetFocus - Example Code

This example gives the client the focus if it does not already have it.

```
#define INCL_WININPUT
#include <OS2.H>
#define SYS_MENU 900
HWND hwndFrame;

if (WinQueryFocus(HWND_DESKTOP) !=          /* returns handle of */
    WinWindowFromID(hwndFrame,FID_CLIENT)) /* window with focus. */
{
    WinSetFocus(HWND_DESKTOP,
    WinWindowFromID(hwndFrame,FID_CLIENT)); /* handle of client */
}
```

WinSetFocus - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinSetHook

WinSetHook - Syntax

This function installs an application procedure into a specified hook chain.

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
HMQ      hmq;          /* Queue identity. */
LONG     lHookType;    /* Hook-chain type. */
PFN      pHookProc;    /* Address of the application hook procedure. */
HMODULE  Module;       /* Resource identity. */
BOOL     rc;           /* Success indicator. */

rc = WinSetHook(hab, hmq, lHookType, pHookProc,
    Module);
```

WinSetHook Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinSetHook Parameter - hmq

hmq ([HMQ](#)) - input
Queue identity.

This parameter identifies the queue to which the hook chain belongs. If *hmq* is set to NULLHANDLE, the hook is installed in the system hook chain. If *hmq* is set to HMQ_CURRENT, the hook is installed in the message queue associated with the current thread (calling thread).

WinSetHook Parameter - IHookType

IHookType ([LONG](#)) - input
Hook-chain type.

- HK_CHECKMSGFILTER
See [CheckMsgFilterHook](#)
- HK_CODEPAGECHANGED
See [CodePageChangedHook](#)
- HK_DESTROYWINDOW
See [DestroyWindowHook](#)
- HK_FINDWORD
See [FindWordHook](#)
- HK_FLUSHBUF
See [FlushBufHook](#)
- HK_HELP
See [HelpHook](#)
- HK_INPUT
See [InputHook](#)
- HK_JOURNALPLAYBACK
See [JournalPlaybackHook](#)
- HK_JOURNALRECORD
See [JournalRecordHook](#)
- HK_LOADER
See [LoaderHook](#)
- HK_LOCKUP
See [LockupHook](#)
- HK_MSGCONTROL
See [MsgControlHook](#)
- HK_MSGFILTER
See [MsgFilterHook](#)
- HK_MSGINPUT
See [MsgInputHook](#)
- HK_PLIST_ENTRY
See [ProgramListEntryHook](#)
- HK_PLIST_EXIT
See [ProgramListExitHook](#)
- HK_REGISTERUSERMSG
See [RegisterUserHook](#)

HK_SENDMSG

See [SendMsgHook](#)

HK_WINDOWDC

See [WindowDCHook](#)

WinSetHook Parameter - pHookProc

pHookProc ([PFN](#)) - input

Address of the application hook procedure.

WinSetHook Parameter - Module

Module ([HMODULE](#)) - input

Resource identity.

Handle of the module that contains the application hook procedure, as returned by the `DosLoadModule` or `DosQueryModuleHandle` call. This parameter can be `NULLHANDLE` when a queue hook is being installed by an application into its own message queue.

When hooking a system hook this parameter must be a valid module handle.

WinSetHook Return Value - rc

rc ([BOOL](#)) - returns

Success indicator.

TRUE

Successful completion

FALSE

An error occurred.

WinSetHook - Parameters

hab ([HAB](#)) - input

Anchor-block handle.

hmq ([HMQ](#)) - input

Queue identity.

This parameter identifies the queue to which the hook chain belongs. If *hmq* is set to `NULLHANDLE`, the hook is installed in the system hook chain. If *hmq* is set to `HMQ_CURRENT`, the hook is installed in the message queue associated with the current thread (calling thread).

IHookType ([LONG](#)) - input

Hook-chain type.

HK_CHECKMSGFILTER
 See [CheckMsgFilterHook](#)
 HK_CODEPAGECHANGED
 See [CodePageChangedHook](#)
 HK_DESTROYWINDOW
 See [DestroyWindowHook](#)
 HK_FINDWORD
 See [FindWordHook](#)
 HK_FLUSHBUF
 See [FlushBufHook](#)
 HK_HELP
 See [HelpHook](#)
 HK_INPUT
 See [InputHook](#)
 HK_JOURNALPLAYBACK
 See [JournalPlaybackHook](#)
 HK_JOURNALRECORD
 See [JournalRecordHook](#)
 HK_LOADER
 See [LoaderHook](#)
 HK_LOCKUP
 See [LockupHook](#)
 HK_MSGCONTROL
 See [MsgControlHook](#)
 HK_MSGFILTER
 See [MsgFilterHook](#)
 HK_MSGINPUT
 See [MsgInputHook](#)
 HK_PLIST_ENTRY
 See [ProgramListEntryHook](#)
 HK_PLIST_EXIT
 See [ProgramListExitHook](#)
 HK_REGISTERUSERMSG
 See [RegisterUserHook](#)
 HK_SENDMSG
 See [SendMsgHook](#)
 HK_WINDOWDC
 See [WindowDCHook](#)

pHookProc ([PFN](#)) - input
 Address of the application hook procedure.

Module ([HMODULE](#)) - input
 Resource identity.

Handle of the module that contains the application hook procedure, as returned by the `DosLoadModule` or `DosQueryModuleHandle` call. This parameter can be `NULLHANDLE` when a queue hook is being installed by an application into its own message queue.

When hooking a system hook this parameter must be a valid module handle.

rc ([BOOL](#)) - returns
 Success indicator.

TRUE Successful completion
 FALSE An error occurred.

WinSetHook - Remarks

Queue hooks are called before system hooks.

This function installs the hook at the *head* of either the system or queue chain. The most recently installed hook is called first.

Use the [WinQueryWindowULong](#) function to obtain the queue handle associated with a window handle.

WinSetHook - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HMQ (0x1002)

An invalid message-queue handle was specified.

PMERR_PARAMETER_OUT_OF_RANGE (0x1003)

The value of a parameter was not within the defined valid range for that parameter.

WinSetHook - Related Functions

Related Functions

- [WinCallMsgFilter](#)
 - [WinReleaseHook](#)
 - [WinSetHook](#)
-

WinSetHook - Example Code

This example uses the WinSetHook call to intercept user-input messages from the application queue.

```
#define INCL_WINHOOKS
#include <OS2.H>
void RecordHook(HAB hab, PQMSG pqmsg); /* prototype of hook */
/* procedure. */

samp()
{
    HAB hab;
    WinSetHook(hab,
        HMQ_CURRENT,
        HK_JOURNALRECORD,
        (PFN)RecordHook,
        (HMODULE)0); /* hook is into application queue. */

    WinReleaseHook(hab,
        HMQ_CURRENT,
        HK_JOURNALRECORD,
        (PFN)RecordHook,
        (HMODULE)0); /* hook is into application queue, */

}
/* This hook records user-input messages. */
void RecordHook(HAB hab, PQMSG pqmsg)
{
    /* ... */
}
```

WinSetHook - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSetKeyboardStateTable

WinSetKeyboardStateTable - Syntax

This function gets or sets the keyboard state.

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND     hwndDesktop; /* Desktop-window handle. */
PBYTE    abKeyStateTable; /* Key state table. */
BOOL     fSet; /* Set indicator. */
BOOL     rc; /* Success indicator. */

rc = WinSetKeyboardStateTable(hwndDesktop,
    abKeyStateTable, fSet);
```

WinSetKeyboardStateTable Parameter - hwndDesktop

hwndDesktop (**HWND**) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

WinSetKeyboardStateTable Parameter - abKeyStateTable

abKeyStateTable (**PBYTE**) - in/out
Key state table.

This is a 256-byte table indexed by virtual key value.

For any virtual key, the 0x80 bit is set if the key is down, and zero if it is up. The 0x01 bit is set if the key is toggled, (pressed an odd number of times), otherwise it is zero.

WinSetKeyboardStateTable Parameter - fSet

fSet (**BOOL**) - input
Set indicator.

TRUE	The keyboard state is set from <i>abKeyStateTable</i>
FALSE	The keyboard state is copied to <i>abKeyStateTable</i> .

WinSetKeyboardStateTable Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetKeyboardStateTable - Parameters

hwndDesktop (**HWND**) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

abKeyStateTable (**PBYTE**) - in/out
Key state table.

This is a 256-byte table indexed by virtual key value.

For any virtual key, the 0x80 bit is set if the key is down, and zero if it is up. The 0x01 bit is set if the key is toggled, (pressed an odd number of times), otherwise it is zero.

fSet (**BOOL**) - input
Set indicator.

TRUE	The keyboard state is set from <i>abKeyStateTable</i>
FALSE	The keyboard state is copied to <i>abKeyStateTable</i> .

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetKeyboardStateTable - Remarks

This function does not change the physical state of the keyboard, but changes the value returned by [WinGetKeyState](#), not [WinGetPhysKeyState](#).

To set the state of a single key, first get the entire table, modify the individual key, and then set the table from the modified value.

WinSetKeyboardStateTable - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinSetKeyboardStateTable - Related Functions

Related Functions

- [WinEnablePhysInput](#)
- [WinFocusChange](#)
- [WinGetKeyState](#)
- [WinGetPhysKeyState](#)
- [WinQueryFocus](#)
- [WinSetFocus](#)
- [WinSetKeyboardStateTable](#)

WinSetKeyboardStateTable - Example Code

This example turns on Caps Lock.

```
#define INCL_WININPUT
#include <OS2.H>
BYTE KeyState[256]; /* This is a 256 byte table */
/* indexed by virtual key */
/* value. */
/* For any virtual key, the */
/* 0x80 bit is set if the key */
/* is down, and zero if it is */
/* up. The 0x01 bit is set */
/* if the key is toggled, */
/* (pressed an odd number */
/* of times), otherwise it is */
/* zero. */
WinSetKeyboardStateTable(HWND_DESKTOP,
```

```

        KeyState,
        FALSE); /* get a copy of the keyboard */
        /* state. */
KeyState[VK_CAPSLOCK] |= 0x01; /* set the CAPSLOCK key to */
        /* on state */
WinSetKeyboardStateTable(HWND_DESKTOP,
        KeyState,
        TRUE); /* set the keyboard state */

```

WinSetKeyboardStateTable - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinSetLboxItemText

WinSetLboxItemText - Syntax

This macro sets the text of the list box indexed item to buffer.

```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndLbox; /* List box handle. */
LONG    index;    /* Index of the list box item. */
PSZ     psz;      /* Pointer to a null terminated string. */
BOOL    rc;       /* Success indicator. */

rc = WinSetLboxItemText(hwndLbox, index, psz);

```

WinSetLboxItemText Parameter - hwndLbox

hwndLbox ([HWND](#)) - input
List box handle.

WinSetLboxItemText Parameter - index

index (**LONG**) - input
Index of the list box item.

WinSetLboxItemText Parameter - psz

psz (**PSZ**) - input
Pointer to a null terminated string.

WinSetLboxItemText Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetLboxItemText - Parameters

hwndLbox (**HWND**) - input
List box handle.

index (**LONG**) - input
Index of the list box item.

psz (**PSZ**) - input
Pointer to a null terminated string.

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetLboxItemText - Remarks

This macro is defined as:

```
#define WinSetLboxItemText(hwndLbox, index, psz) \
    ((BOOL)WinSendMsg(hwndLbox, \
        LM_SETITEMTEXT, \
        MPFROMLONG(index), \
        MPFROMP(psz)))
```

This macro requires the existence of a message queue.

WinSetLboxItemText - Related Functions

Related Functions

- [WinSendMsg](#)
-

WinSetLboxItemText - Related Messages

Related Messages

- [LM_SETITEMTEXT](#)
-

WinSetLboxItemText - Example Code

This example uses the WinSetLboxItemText call to set the months in a calendar list box.

```
#define INCL_WINDIALOGS
#include <OS2.H>

HWND    hwndLbox;
LONG    i;
typedef char MONTH[12];
MONTH months[12] = {"January", "February", "March",
    "April", "May", "June", "July",
    "August", "September", "October",
    "November", "December" };

for (i=0; i<12; i++)
{
    WinSetLboxItemText(hwndLbox,
        i,
        months[i]);
} /* endfor */
```

WinSetLboxItemText - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Related Messages](#)

[Glossary](#)

WinSetMenuItemText

WinSetMenuItemText - Syntax

This macro sets the text for Menu indexed item to buffer.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndMenu; /* Menu window handle. */
USHORT     usId;     /* Identity of the menu item. */
PSZ        pText;    /* Text for the menu item. */
BOOL       rc;        /* Success indicator. */

rc = WinSetMenuItemText(hwndMenu, usId, pText);
```

WinSetMenuItemText Parameter - hwndMenu

hwndMenu ([HWND](#)) - input
Menu window handle.

WinSetMenuItemText Parameter - usId

usId ([USHORT](#)) - input
Identity of the menu item.

WinSetMenuItemText Parameter - pText

pText ([PSZ](#)) - input
Text for the menu item.

WinSetMenuItemText Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetMenuItemText - Parameters

hwndMenu ([HWND](#)) - input
Menu window handle.

usId ([USHORT](#)) - input
Identity of the menu item.

pText ([PSZ](#)) - input
Text for the menu item.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetMenuItemText - Remarks

This macro expands to:

```
#define WinSetMenuItemText (hwndMenu, usId, pszText)
( (BOOL)WinSendMsg(hwndMenu,
    MM_SETITEMTEXT,
    MPFROMSHORT(id),
    MPFROMP(pszText)) )
```

This function requires the existence of a message queue.

WinSetMenuItemText - Related Functions

Related Functions

- [WinSendMsg](#)
-

WinSetMenuItemText - Related Messages

Related Messages

- [MM_SETITEMTEXT](#)
-

WinSetMenuItemText - Example Code

This example sets the options text in a menu.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
#define IDM_OPTIONS 900

HWND      hwndMenu;

WinSetMenuItemText(hwndMenu,
                    IDM_OPTIONS,
                    "Options");
```

WinSetMenuItemText - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinSetMsgInterest

WinSetMsgInterest - Syntax

This function sets a window's message interest.

```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND     hwnd;          /* Window handle. */
ULONG    ulMsgClass;    /* Message class to have interest level set. */
LONG     lControl;      /* Interest-identifier for the message class. */
BOOL     rc;            /* Interest-changed indicator. */

rc = WinSetMsgInterest(hwnd, ulMsgClass, lControl);
```

WinSetMsgInterest Parameter - hwnd

hwnd (**HWND**) - input
Window handle.

WinSetMsgInterest Parameter - ulMsgClass

ulMsgClass (**ULONG**) - input
Message class to have interest level set.

msgid	A single message identity (for example, WM_SHOW)
SMIM_ALL	All messages (except for WM_QUIT if <i>lControl</i> is SMI_AUTODISPATCH or SMI_NOINTEREST).

WinSetMsgInterest Parameter - lControl

lControl (**LONG**) - input
Interest-identifier for the message class.

SMI_RESET	Revert to interest specified for the window class.
SMI_INTEREST	Interested in the messages.
SMI_NOINTEREST	Not interested in the messages.
SMI_AUTODISPATCH	Interested in the message or messages, but they are to be automatically dispatched to the window procedure.

WinSetMsgInterest Return Value - rc

rc ([BOOL](#)) - returns
Interest-changed indicator.

TRUE	Interest successfully changed
FALSE	Interest not successfully changed.

WinSetMsgInterest - Parameters

hwnd ([HWND](#)) - input
Window handle.

ulMsgClass ([ULONG](#)) - input
Message class to have interest level set.

msgid	A single message identity (for example, WM_SHOW)
SMIM_ALL	All messages (except for WM_QUIT if <i>lControl</i> is SMI_AUTODISPATCH or SMI_NOINTEREST).

lControl ([LONG](#)) - input
Interest-identifier for the message class.

SMI_RESET	Revert to interest specified for the window class.
SMI_INTEREST	Interested in the messages.
SMI_NOINTEREST	Not interested in the messages.
SMI_AUTODISPATCH	Interested in the message or messages, but they are to be automatically dispatched to the window procedure.

rc ([BOOL](#)) - returns
Interest-changed indicator.

TRUE	Interest successfully changed
FALSE	Interest not successfully changed.

WinSetMsgInterest - Remarks

This function has no effect unless the [MsgControlHook](#) hook, which is invoked by this function, has been set. The interest for WM_QUIT cannot be set to SMI_AUTODISPATCH using SMIM_ALL, because WM_QUIT is the normal means of terminating an application. It can be set specifically, if required.

WinSetMsgInterest - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinSetMsgInterest - Related Functions

Related Functions

- [WinBroadcastMsg](#)
- [WinCreateMsgQueue](#)
- [WinDestroyMsgQueue](#)
- [WinDispatchMsg](#)
- [WinGetDlgMsg](#)
- [WinGetMsg](#)
- [WinInSendMessage](#)
- [WinPeekMsg](#)
- [WinPostMsg](#)
- [WinPostQueueMsg](#)
- [WinQueryMsgPos](#)
- [WinQueryMsgTime](#)
- [WinQueryQueueInfo](#)
- [WinQueryQueueStatus](#)
- [WinSendDlgItemMsg](#)
- [WinSendMessage](#)
- [WinSetClassMsgInterest](#)
- [WinSetMsgInterest](#)
- [WinSetMsgMode](#)
- [WinSetSynchroMode](#)
- [WinWaitMsg](#)

WinSetMsgInterest - Example Code

This example uses the WinSetMsgInterest call to set the message interest of a window to only WM_SHOW messages.

```
#define INCL_WINMESSAGEGR
#define INCL_WINHOOKS
#include <OS2.H>

HWND hwnd;
HAB hab;
BOOL fSuccess;
/* Hook Procedure Prototype */

BOOL MsgCtlHook(HAB hab, LONG idContext, /* this hook can */
                HWND hwnd, PSZ pszClassname, /* be given any */
                ULONG ulMsgclass, /* name. */
                LONG idControl, PBOOL fSuccess);

main()
{
/* This function passes the hook procedure address to the system. */

WinSetHook(hab,
            (HMQ)0,
            MCHK_CLASSMSGINTEREST,
            (PFN)MsgCtlHook,
            (HMODULE)0); /* hook is into application queue. */

/* This function sets the message interest of a window class. */
WinSetMsgInterest(hwnd,
```

```

        WM_SHOW,
        SMI_AUTODISPATCH); /* interested in the */
        /* messages, but they are to */
        /* be automatically dispatched */
        /* to the window procedure. */
    }

    BOOL MsgCtlHook(HAB hab, LONG idContext,          /* this hook can */
                    HWND hwnd, PSZ pszClassname,      /* be given any */
                    ULONG ulMsgclass,                 /* name. */
                    LONG idControl, PBOOL fSuccess)
{
    /* ... */
}

```

WinSetMsgInterest - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSetMsgMode

WinSetMsgMode - Syntax

This function indicates the mode for the generation and processing of messages for the private window class of an application.

```

#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HAB      hab;          /* Anchor block handle. */
PSZ      pszClassName; /* Window class name. */
LONG     lControl;     /* Message mode identifier. */
BOOL     rc;           /* Message delay indicator. */

rc = WinSetMsgMode(hab, pszClassName, lControl);

```

WinSetMsgMode Parameter - hab

hab (**HAB**) - input
Anchor block handle.

WinSetMsgMode Parameter - pszClassName

pszClassName (**PSZ**) - input
Window class name.

WinSetMsgMode Parameter - IControl

IControl (**LONG**) - input
Message mode identifier.

SMD_DELAYED The generation of messages may be delayed

SMD_IMMEDIATE The generation of messages will not be delayed.

WinSetMsgMode Return Value - rc

rc (**BOOL**) - returns
Message delay indicator.

TRUE Message mode successfully set

FALSE Message mode not successfully set.

WinSetMsgMode - Parameters

hab (**HAB**) - input
Anchor block handle.

pszClassName (**PSZ**) - input
Window class name.

IControl (**LONG**) - input
Message mode identifier.

SMD_DELAYED The generation of messages may be delayed

SMD_IMMEDIATE The generation of messages will not be delayed.

rc ([BOOL](#)) - returns
Message delay indicator.

TRUE	Message mode successfully set
FALSE	Message mode not successfully set.

WinSetMsgMode - Remarks

This function has no effect unless the [MsgControlHook](#) hook, which is invoked by this function, has been set.

WinSetMsgMode - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinSetMsgMode - Related Functions

Related Functions

- [WinBroadcastMsg](#)
 - [WinCreateMsgQueue](#)
 - [WinDestroyMsgQueue](#)
 - [WinDispatchMsg](#)
 - [WinGetDlgMsg](#)
 - [WinGetMsg](#)
 - [WinInSendMessage](#)
 - [WinPeekMsg](#)
 - [WinPostMsg](#)
 - [WinPostQueueMsg](#)
 - [WinQueryMsgPos](#)
 - [WinQueryMsgTime](#)
 - [WinQueryQueueInfo](#)
 - [WinQueryQueueStatus](#)
 - [WinSendDlgItemMsg](#)
 - [WinSendMessage](#)
 - [WinSetClassMsgInterest](#)
 - [WinSetMsgInterest](#)
 - [WinSetMsgMode](#)
 - [WinSetSynchronMode](#)
 - [WinWaitMsg](#)
-

WinSetMsgMode - Example Code

This example uses the `WinSetMsgMode` call to set the a delayed message processing mode for private window class "Generic".

```

#define INCL_WINWINDOWMGR
#define INCL_WINMESSAGEMGR
#include <OS2.H>
HWND hwnd;
HAB hab;
PFNWP GenericWndProc;
CHAR szClassName[] = "Generic"; /* window class name */

if (!WinRegisterClass(hab, /* anchor-block handle */
    szClassName, /* class name */
    GenericWndProc, /* window procedure */
    0L, /* window style */
    0)); /* amount of reserved memory */
    return (FALSE);

WinSetMsgMode(hab,
    "Generic",
    SMD_DELAYED);

```

WinSetMsgMode - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSetMultWindowPos

WinSetMultWindowPos - Syntax

This function performs the [WinSetWindowPos](#) function for *cswp* windows, using *pswp*, an array of structures whose elements correspond to the input parameters of [WinSetWindowPos](#).

```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab; /* Anchor-block handle. */
PSWP     pswp; /* An array of set window position (SWP) structures. */
ULONG    cswp; /* Window count. */
BOOL     rc; /* Positioning success indicator. */

rc = WinSetMultWindowPos(hab, pswp, cswp);

```

WinSetMultWindowPos Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinSetMultWindowPos Parameter - pswp

pswp ([PSWP](#)) - input
An array of set window position ([SWP](#)) structures.

The elements of each correspond to the input parameters of [WinSetWindowPos](#).

WinSetMultWindowPos Parameter - cswp

cswp ([ULONG](#)) - input
Window count.

This parameter can be set to 0, representing an empty list of [SWP](#) structures.

WinSetMultWindowPos Return Value - rc

rc ([BOOL](#)) - returns
Positioning success indicator.

TRUE	Positioning succeeded
FALSE	Positioning failed.

WinSetMultWindowPos - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

pswp ([PSWP](#)) - input
An array of set window position ([SWP](#)) structures.

The elements of each correspond to the input parameters of [WinSetWindowPos](#).

cswp ([ULONG](#)) - input
Window count.

This parameter can be set to 0, representing an empty list of [SWP](#) structures.

rc ([BOOL](#)) - returns
Positioning success indicator.

TRUE	Positioning succeeded
FALSE	Positioning failed.

WinSetMultWindowPos - Remarks

All windows being positioned must have the same parent.

It is more efficient to use this function than to issue multiple [WinSetWindowPos](#) functions, as it causes less screen updating. If specifies a frame window, this function recalculates the sizes and positions of the frame controls. If the new window rectangle for any frame control is to be empty, instead of resizing or repositioning that control, it is hidden by (SWP_HIDE) instead. This eliminates needless processing of windows that are not visible. The window rectangle of the control in question is left in its original state. For example, if [WinSetWindowPos](#) is issued to change the size of a standard frame window to an empty rectangle, and [WinQueryWindowRect](#) is issued against the client window, the rectangle returned is not an empty rectangle, but the original client rectangle before [WinSetWindowPos](#) was issued.

WinSetMultWindowPos - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_INVALID_FLAG (0x1019)
An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

WinSetMultWindowPos - Related Functions

Related Functions

- [WinGetMinPosition](#)
- [WinQueryActiveWindow](#)
- [WinQueryWindowPos](#)
- [WinSaveWindowPos](#)
- [WinSetActiveWindow](#)
- [WinSetMultWindowPos](#)
- [WinSetWindowPos](#)

WinSetMultWindowPos - Related Messages

Related Messages

- [WM_ACTIVATE](#)
- [WM_ADJUSTWINDOWPOS](#)
- [WM_CALCVALIDRECTS](#)

- [WM_MOVE](#)
- [WM_SHOW](#)
- [WM_SIZE](#)

WinSetMultWindowPos - Example Code

This example uses the WinSetMultWindowPos to cascade up to 16 main windows.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HWND ahwnd[16]; /* Array of window handles */
SWP aSwp[16]; /* Array of SWP structures */
SWP swp;
HAB hab;
LONG xcoord, ycoord;
LONG i=1;

/* Get the recommended window position */
WinQueryTaskSizePos(hab,
                    0,
                    &swp);

xcoord = swp.x;
ycoord = swp.y;

/* Initialize the array of SWP structures */
/* where each is displaced by (10,10) */
for (i=0; i<16 ; i++ )
{
    aSwp[i] = swp;
    aSwp[i].x = xcoord;
    aSwp[i].y = ycoord;
    xcoord += 10;
    ycoord += 10;
} /* endfor */

/* Get a list of all the main windows into the ahwnd array */
i=0
henum = WinBeginEnumWindows(HWND_DESKTOP);
do
{
    ahwnd[i] = WinGetNextWindow(henum);
}
while((ahwnd[i++]!= NULL) && i < 16);
WinEndEnumWindows(henum);
WinSetMultWindowPos(hab,aSwp,i-1);
```

WinSetMultWindowPos - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinSetObjectData

WinSetObjectData - Syntax

The WinSetObjectData function is called to set data on a workplace object.

```
#define INCL_WINWORKPLACE
#include <os2.h>

HOBJECT    object;        /* Handle to a workplace object. */
PSZ        pSetupString;  /* Pointer to object-specific parameters. */
BOOL       rc;            /* Success indicator. */

rc = WinSetObjectData(object, pSetupString);
```

WinSetObjectData Parameter - object

object ([HOBJECT](#)) - input
Handle to a workplace object.

WinSetObjectData Parameter - pSetupString

pSetupString ([PSZ](#)) - input
Pointer to object-specific parameters.

A pointer to a zero-terminated string which contains the object-specific parameters to the new object.

The *pSetupString* string is extracted when the wpSetup method is called. For a listing of setup strings, see individual object classes; See WPLaunchPad class definition for a discussion of setup string values used in Toolbar customization.

WinSetObjectData Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE
Successful completion.

FALSE

Error occurred.

WinSetObjectData - Parameters

object ([HOBJECT](#)) - input
Handle to a workplace object.

pSetupString ([PSZ](#)) - input
Pointer to object-specific parameters.

A pointer to a zero-terminated string which contains the object-specific parameters to the new object.

The *pSetupString* string is extracted when the wpSetup method is called. For a listing of setup strings, see individual object classes; See WPLaunchPad class definition for a discussion of setup string values used in Toolbar customization.

rc ([BOOL](#)) - returns
Success indicator.

TRUE
Successful completion.

FALSE
Error occurred.

WinSetObjectData - Remarks

The WinSetObjectData function will change settings on an object that was created with the WinCreateObject function.

Using [HOBJECT](#) for .INI files or files in which an application uses a rename/save/delete sequence is not supported.

WinSetObjectData - Related Functions

Related Functions

- [WinCreateObject](#)
- [WinSetObjectData](#)
- [WinDestroyObject](#)

WinSetObjectData - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Related Functions](#)

[Glossary](#)

WinSetOwner

WinSetOwner - Syntax

This function changes the owner window of a specified window.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd;           /* Window handle whose owner window is to be changed. */
HWND    hwndNewOwner;   /* Handle of the new owner. */
BOOL    rc;             /* Success indicator. */

rc = WinSetOwner(hwnd, hwndNewOwner);
```

WinSetOwner Parameter - hwnd

hwnd (**HWND**) - input
Window handle whose owner window is to be changed.

WinSetOwner Parameter - hwndNewOwner

hwndNewOwner (**HWND**) - input
Handle of the new owner.

NULLHANDLE	The window becomes "disowned"
Other	Handle of the new owner window.

WinSetOwner Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
------	-----------------------

FALSE

Error occurred.

WinSetOwner - Parameters

hwnd ([HWND](#)) - input

Window handle whose owner window is to be changed.

hwndNewOwner ([HWND](#)) - input

Handle of the new owner.

NULLHANDLE

The window becomes "disowned"

Other

Handle of the new owner window.

rc ([BOOL](#)) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinSetOwner - Remarks

The old owner window is not locked by this function.

The [WinQueryWindow](#) function can be used to get the handle of the owner window.

WinSetOwner - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)

An invalid window handle was specified.

WinSetOwner - Related Functions

Related Functions

- [WinBeginEnumWindows](#)
- [WinEndEnumWindows](#)
- [WinEnumDlgItem](#)
- [WinGetNextWindow](#)
- [WinIsChild](#)
- [WinMultWindowFromIDs](#)
- [WinQueryWindow](#)

- [WinSetOwner](#)
- [WinSetParent](#)

WinSetOwner - Example Code

This example uses the WinSetOwner call to "disown" a window.

```
#define INCL_WINWINDOWMGR

#include <OS2.H>

HWND hwnd;          /* window handles. */
WinSetOwner(hwnd, (HWND)0);
```

WinSetOwner - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSetParent

WinSetParent - Syntax

This function sets the parent for *hwnd* to *hwndNewParent*.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd;          /* Window handle. */
HWND    hwndNewParent; /* New parent window handle. */
BOOL    fRedraw;        /* Redraw indicator. */
BOOL    rc;             /* Parent-changed indicator. */

rc = WinSetParent(hwnd, hwndNewParent, fRedraw);
```

WinSetParent Parameter - hwnd

hwnd ([HWND](#)) - input
Window handle.

WinSetParent Parameter - hwndNewParent

hwndNewParent ([HWND](#)) - input
New parent window handle.

- This cannot be a descendant of *hwnd*.
- If this parameter is a desktop window handle or `HWND_DESKTOP`, *hwnd* becomes a main window.
- If this parameter is not equal to `HWND_OBJECT`, it must be a descendant of the same desktop window as *hwnd*.
- If this parameter is `HWND_OBJECT` or a window handle returned by [WinQueryObjectWindow](#), *hwnd* becomes an object window.

WinSetParent Parameter - fRedraw

fRedraw ([BOOL](#)) - input
Redraw indicator.

- TRUE**
If *hwnd* is visible, any necessary redrawing of both the old parent and the new parent windows is performed.
- FALSE**
No redrawing of the old and new parent windows is performed. This avoids an extra device update when subsequent calls cause the windows to be redrawn.

WinSetParent Return Value - rc

rc ([BOOL](#)) - returns
Parent-changed indicator.

- TRUE**
Parent successfully changed
- FALSE**
Parent not successfully changed.

WinSetParent - Parameters

hwnd ([HWND](#)) - input
Window handle.

hwndNewParent ([HWND](#)) - input
New parent window handle.

This cannot be a descendant of *hwnd*.

If this parameter is a desktop window handle or `HWND_DESKTOP`, *hwnd* becomes a main window.

If this parameter is not equal to `HWND_OBJECT`, it must be a descendant of the same desktop window as *hwnd*.

If this parameter is `HWND_OBJECT` or a window handle returned by [WinQueryObjectWindow](#), *hwnd* becomes an object window.

fRedraw ([BOOL](#)) - input
Redraw indicator.

TRUE
If *hwnd* is visible, any necessary redrawing of both the old parent and the new parent windows is performed.

FALSE
No redrawing of the old and new parent windows is performed. This avoids an extra device update when subsequent calls cause the windows to be redrawn.

rc ([BOOL](#)) - returns
Parent-changed indicator.

TRUE
Parent successfully changed

FALSE
Parent not successfully changed.

WinSetParent - Remarks

This function returns an error (`PMERR_INVALID_HWND`) if the user specifies the desktop window as the new parent of a window to which a clip region is currently set.

WinSetParent - Errors

Possible returns from [WinGetLastError](#)

`PMERR_INVALID_HWND` (0x1001)
An invalid window handle was specified.

`PMERR_INVALID_FLAG` (0x1019)
An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

WinSetParent - Related Functions

- Related Functions**
- [WinBeginEnumWindows](#)

- [WinEndEnumWindows](#)
- [WinEnumDlgItem](#)
- [WinGetNextWindow](#)
- [WinIsChild](#)
- [WinMultWindowFromIDs](#)
- [WinQueryWindow](#)
- [WinSetClipRegion](#)
- [WinSetOwner](#)

WinSetParent - Related Messages

Related Messages

- [WM_PAINT](#)

WinSetParent - Example Code

This example uses the WinSetParent call to change a window to a main window.

```
#define INCL_WINWINDOWMGR

#include <OS2.H>

HWND hwnd;          /* Window handles */
WinSetParent(hwnd,
              HWND_DESKTOP,
              TRUE); /* Do any necessary redrawing */
```

WinSetParent - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinSetPointer

WinSetPointer - Syntax

This call sets the desktop-pointer handle.

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndDesktop;    /* Desktop-window handle. */
HPOINTER  hptrNewPointer; /* New pointer handle. */
BOOL      rc;             /* Pointer-updated indicator. */

rc = WinSetPointer(hwndDesktop, hptrNewPointer);
```

WinSetPointer Parameter - hwndDesktop

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

WinSetPointer Parameter - hptrNewPointer

hptrNewPointer ([HPOINTER](#)) - input
New pointer handle.

NULL	Remove pointer from the screen.
Other	Pointer handle associated with <i>hwndDesktop</i> . Handles for application-defined pointers are returned by the WinLoadPointer and WinCreatePointer calls.

WinSetPointer Return Value - rc

rc ([BOOL](#)) - returns
Pointer-updated indicator.

TRUE	Pointer successfully updated
FALSE	Pointer not successfully updated.

WinSetPointer - Parameters

hwndDeskTop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP
The desktop-window handle
Other
Specified desktop-window handle.

hptrNewPointer ([HPOINTER](#)) - input
New pointer handle.

NULL
Remove pointer from the screen.
Other
Pointer handle associated with *hwndDeskTop*. Handles for application-defined pointers are returned by the [WinLoadPointer](#) and [WinCreatePointer](#) calls.

rc ([BOOL](#)) - returns
Pointer-updated indicator.

TRUE
Pointer successfully updated
FALSE
Pointer not successfully updated.

WinSetPointer - Remarks

This call is very efficient if *hptrNewPointer* is the same as the current pointer handle.

WinSetPointer - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_INVALID_HPTR (0x101B)
An invalid pointer handle was specified.

PMERR_INV_CURSOR_BITMAP (0x205E)
An invalid pointer was referenced with WinSetPointer.

WinSetPointer - Related Functions

Related Functions

- [WinCreatePointer](#)
- [WinCreatePointerIndirect](#)

- [WinDestroyPointer](#)
- [WinDrawPointer](#)
- [WinLoadPointer](#)
- [WinQueryPointer](#)
- [WinQueryPointerInfo](#)
- [WinQueryPointerPos](#)
- [WinQuerySysPointer](#)
- [WinQuerySysPointerData](#)
- [WinSetPointer](#)
- [WinSetPointerPos](#)
- [WinSetSysPointerData](#)
- [WinShowPointer](#)

WinSetPointer - Example Code

This example calls WinLoadPointer to load an application-defined pointer. When processing the WM_MOUSEMOVE message, the loaded pointer is displayed by calling WinSetPointer.

```
#define INCL_WININPUT
#define INCL_WINPOINTERS
#include <OS2.H>
#define IDP_CROSSHAIR 900

HPOINTER hptrCrossHair;
USHORT msg;

switch(msg)
{
case WM_CREATE:
    hptrCrossHair = WinLoadPointer(HWND_DESKTOP,
                                  (ULONG)0, /* load from .exe file */
                                  IDP_CROSSHAIR); /* identifies the pointer */

case WM_MOUSEMOVE:
    WinSetPointer(HWND_DESKTOP, hptrCrossHair);
}
```

WinSetPointer - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Errors](#)
- [Remarks](#)
- [Example Code](#)
- [Related Functions](#)
- [Glossary](#)

WinSetPointerOwner

WinSetPointerOwner - Syntax

This function is specific to OS/2 Version 2.1 or higher.

This function allows an application to declare that a pointer it created can now be used by ANY process in the system (not just the process that created the pointer).

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HPOINTER  hptr;      /* Handle of pointer. */
PID        pid;      /* Process identity. */
BOOL       fDestroy; /* Pointer destruction flag. */
BOOL       rc;        /* Success indicator. */

rc = WinSetPointerOwner(hptr, pid, fDestroy);
```

WinSetPointerOwner Parameter - hptr

hptr (**HPOINTER**) - input
Handle of pointer.

WinSetPointerOwner Parameter - pid

pid (**PID**) - input
Process identity.

WinSetPointerOwner Parameter - fDestroy

fDestroy (**BOOL**) - input
Pointer destruction flag.

TRUE	Pointer can be destroyed by the current owner.
FALSE	Pointer cannot be destroyed by any owner.

WinSetPointerOwner Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Pointer owner successfully updated.
FALSE	Pointer owner not successfully updated.

WinSetPointerOwner - Parameters

hptr ([HPOINTER](#)) - input
Handle of pointer.

pid ([PID](#)) - input
Process identity.

fDestroy ([BOOL](#)) - input
Pointer destruction flag.

TRUE	Pointer can be destroyed by the current owner.
FALSE	Pointer cannot be destroyed by any owner.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Pointer owner successfully updated.
FALSE	Pointer owner not successfully updated.

WinSetPointerOwner - Remarks

Setting *pid* to 0 actually makes the pointer global (available on all processes).

Setting *pid* to 0, or setting *fDestroy* to FALSE should be done with extreme care. The resources allocated with this pointer will never be freed by the system, unless the pointer is set back to an active process, and *fDestroy* is set to TRUE.

WinSetPointerOwner - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HPTR (0x101B)
An invalid pointer handle was specified.

WinSetPointerOwner - Related Functions

Related Functions

- [WinCreatePointer](#)
- [WinCreatePointerIndirect](#)
- [WinDestroyPointer](#)
- [WinDrawPointer](#)
- [WinLoadPointer](#)
- [WinQueryPointer](#)
- [WinQueryPointerInfo](#)
- [WinQueryPointerPos](#)
- [WinQuerySysPointer](#)
- [WinQuerySysPointerData](#)
- [WinSetPointer](#)
- [WinSetPointerPos](#)
- [WinSetSysPointerData](#)
- [WinShowPointer](#)

WinSetPointerOwner - Example Code

This example makes a pointer global, allowing its use by other processes.

```
WinSetPointerOwner(hMyPtr, 0L, FALSE);
```

On Exit of a DLL/Application using global pointers, the following sample code frees the resources allocated by the system.

```
WinSetPointerOwner(hMyPtr, CurPid, TRUE);  
WinDestroyPointer(hMyPtr);
```

WinSetPointerOwner - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Errors](#)
- [Remarks](#)
- [Example Code](#)
- [Related Functions](#)
- [Glossary](#)

WinSetPointerPos

WinSetPointerPos - Syntax

This function sets the pointer position.

```

#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND     hwndDesktop; /* Desktop-window handle. */
LONG     lx;          /* x-position of pointer in screen coordinates. */
LONG     ly;          /* y-position of pointer in screen coordinates. */
BOOL     rc;          /* Pointer position updated indicator. */

rc = WinSetPointerPos(hwndDesktop, lx, ly);

```

WinSetPointerPos Parameter - hwndDesktop

hwndDesktop (**HWND**) - input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

WinSetPointerPos Parameter - lx

lx (**LONG**) - input
x-position of pointer in screen coordinates.

WinSetPointerPos Parameter - ly

ly (**LONG**) - input
y-position of pointer in screen coordinates.

WinSetPointerPos Return Value - rc

rc (**BOOL**) - returns
Pointer position updated indicator.

TRUE	Pointer position successfully updated
FALSE	Pointer position not successfully updated.

WinSetPointerPos - Parameters

hwndDesktop ([HWND](#)) - input

Desktop-window handle.

HWND_DESKTOP

The desktop-window handle

Other

Specified desktop-window handle.

lx ([LONG](#)) - input

x-position of pointer in screen coordinates.

ly ([LONG](#)) - input

y-position of pointer in screen coordinates.

rc ([BOOL](#)) - returns

Pointer position updated indicator.

TRUE

Pointer position successfully updated

FALSE

Pointer position not successfully updated.

WinSetPointerPos - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)

An invalid window handle was specified.

WinSetPointerPos - Related Functions

Related Functions

- [WinCreatePointer](#)
- [WinCreatePointerIndirect](#)
- [WinDestroyPointer](#)
- [WinDrawPointer](#)
- [WinLoadPointer](#)
- [WinQueryPointer](#)
- [WinQueryPointerInfo](#)
- [WinQueryPointerPos](#)
- [WinQuerySysPointer](#)
- [WinQuerySysPointerData](#)
- [WinSetPointer](#)
- [WinSetPointerPos](#)
- [WinSetSysPointerData](#)
- [WinShowPointer](#)

WinSetPointerPos - Example Code

This example calls WinSetPointer to set the pointer at 50, 50 in Screen coordinates.

```
#define INCL_WINPOINTERS
#include <OS2.H>

WinSetPointerPos(HWND_DESKTOP,
                 (LONG)50,
                 (LONG)50);
```

WinSetPointerPos - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Errors](#)
- [Example Code](#)
- [Related Functions](#)
- [Glossary](#)

WinSetPresParam

WinSetPresParam - Syntax

This function sets a presentation parameter for a window.

```
#define INCL_WINSYS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwnd;          /* Window handle. */
ULONG     idAttrType;     /* Type of presentation parameter attribute to be set. */
ULONG     cbAttrValueLen; /* Size of the buffer pointed to by the pAttrValue parameter. */
PVOID     pAttrValue;     /* Pointer to the presentation parameter's data. */
BOOL      rc;             /* Success indicator. */

rc = WinSetPresParam(hwnd, idAttrType, cbAttrValueLen,
                     pAttrValue);
```

WinSetPresParam Parameter - hwnd

hwnd (**HWND**) - input
Window handle.

This is the window to which changes of the presentation parameters are to be applied.

WinSetPresParam Parameter - idAttrType

idAttrType ([ULONG](#)) - input

Type of presentation parameter attribute to be set.

This parameter can be set to one of the following system-defined presentation parameter attribute types or an application-defined type.

PP_FOREGROUND_COLOR

Foreground color (in [RGB](#)) attribute.

PP_BACKGROUND_COLOR

Background color (in [RGB](#)) attribute.

PP_FOREGROUND_COLOR_INDEX

Foreground color index attribute.

PP_BACKGROUND_COLOR_INDEX

Background color index attribute.

PP_HILITE_FOREGROUND_COLOR

Highlighted foreground color (in [RGB](#)) attribute, for example for selected menu items.

PP_HILITE_BACKGROUND_COLOR

Highlighted background color (in [RGB](#)) attribute.

PP_HILITE_FOREGROUND_COLOR_INDEX

Highlighted foreground color index attribute.

PP_HILITE_BACKGROUND_COLOR_INDEX

Highlighted background color index attribute.

PP_DISABLED_FOREGROUND_COLOR

Disabled foreground color (in [RGB](#)) attribute.

PP_DISABLED_BACKGROUND_COLOR

Disabled background color (in [RGB](#)) attribute.

PP_DISABLED_FOREGROUND_COLOR_INDEX

Disabled foreground color index attribute.

PP_DISABLED_BACKGROUND_COLOR_INDEX

Disabled background color index attribute.

PP_BORDER_COLOR

Border color (in [RGB](#)) attribute.

PP_BORDER_COLOR_INDEX

Border color index attribute.

PP_FONT_NAME_SIZE

Font name and size attribute.

PP_ACTIVE_COLOR

Active color value of data type [RGB](#).

PP_ACTIVE_COLOR_INDEX

Active color index value of data type [LONG](#).

PP_INACTIVE_COLOR

Inactive color value of data type [RGB](#).

PP_INACTIVECOLORINDEX	Inactive color index value of data type LONG .
PP_ACTIVETEXTFGNDCOLOR	Active text foreground color value of data type RGB .
PP_ACTIVETEXTFGNDCOLORINDEX	Active text foreground color index value of data type LONG .
PP_ACTIVETEXTBGNDCOLOR	Active text background color value of data type RGB .
PP_ACTIVETEXTBGNDCOLORINDEX	Active text background color index value of data type LONG .
PP_INACTIVETEXTFGNDCOLOR	Inactive text foreground color value of data type RGB .
PP_INACTIVETEXTFGNDCOLORINDEX	Inactive text foreground color index value of data type LONG .
PP_INACTIVETEXTBGNDCOLOR	Inactive text background color value of data type RGB .
PP_INACTIVETEXTBGNDCOLORINDEX	Inactive text background color index value of data type LONG .
PP_SHADOW	Changes the color used for drop shadows on certain controls.
PP_USER	This is a user-defined presentation parameter.

WinSetPresParam Parameter - cbAttrValueLen

cbAttrValueLen ([ULONG](#)) - input
Size of the buffer pointed to by the *pAttrValue* parameter.

WinSetPresParam Parameter - pAttrValue

pAttrValue ([PVOID](#)) - input
Pointer to the presentation parameter's data.

WinSetPresParam Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	

Error occurred.

WinSetPresParam - Parameters

hwnd ([HWND](#)) - input
Window handle.

This is the window to which changes of the presentation parameters are to be applied.

idAttrType ([ULONG](#)) - input
Type of presentation parameter attribute to be set.

This parameter can be set to one of the following system-defined presentation parameter attribute types or an application-defined type.

PP_FOREGROUNDCOLOR
Foreground color (in [RGB](#)) attribute.

PP_BACKGROUNDCOLOR
Background color (in [RGB](#)) attribute.

PP_FOREGROUNDCOLORINDEX
Foreground color index attribute.

PP_BACKGROUNDCOLORINDEX
Background color index attribute.

PP_HILITEFOREGROUNDCOLOR
Highlighted foreground color (in [RGB](#)) attribute, for example for selected menu items.

PP_HILITEBACKGROUNDCOLOR
Highlighted background color (in [RGB](#)) attribute.

PP_HILITEFOREGROUNDCOLORINDEX
Highlighted foreground color index attribute.

PP_HILITEBACKGROUNDCOLORINDEX
Highlighted background color index attribute.

PP_DISABLEDFOREGROUNDCOLOR
Disabled foreground color (in [RGB](#)) attribute.

PP_DISABLEDBACKGROUNDCOLOR
Disabled background color (in [RGB](#)) attribute.

PP_DISABLEDFOREGROUNDCOLORINDEX
Disabled foreground color index attribute.

PP_DISABLEDBACKGROUNDCOLORINDEX
Disabled background color index attribute.

PP_BORDERCOLOR
Border color (in [RGB](#)) attribute.

PP_BORDERCOLORINDEX
Border color index attribute.

PP_FONTNAMESIZE
Font name and size attribute.

PP_ACTIVECOLOR
Active color value of data type [RGB](#).

PP_ACTIVECOLORINDEX
Active color index value of data type [LONG](#).

PP_INACTIVECOLOR	Inactive color value of data type RGB .
PP_INACTIVECOLORINDEX	Inactive color index value of data type LONG .
PP_ACTIVETEXTFGNDCOLOR	Active text foreground color value of data type RGB .
PP_ACTIVETEXTFGNDCOLORINDEX	Active text foreground color index value of data type LONG .
PP_ACTIVETEXTBGNDCOLOR	Active text background color value of data type RGB .
PP_ACTIVETEXTBGNDCOLORINDEX	Active text background color index value of data type LONG .
PP_INACTIVETEXTFGNDCOLOR	Inactive text foreground color value of data type RGB .
PP_INACTIVETEXTFGNDCOLORINDEX	Inactive text foreground color index value of data type LONG .
PP_INACTIVETEXTBGNDCOLOR	Inactive text background color value of data type RGB .
PP_INACTIVETEXTBGNDCOLORINDEX	Inactive text background color index value of data type LONG .
PP_SHADOW	Changes the color used for drop shadows on certain controls.
PP_USER	This is a user-defined presentation parameter.

cbAttrValueLen ([ULONG](#)) - input
Size of the buffer pointed to by the *pAttrValue* parameter.

pAttrValue ([PVOID](#)) - input
Pointer to the presentation parameter's data.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetPresParam - Remarks

This function associates the presentation parameter attribute identified by *idAttrType* with the window *hwnd*. If the attribute already exists for the window, its value is changed to the new value specified by *pAttrValue*. If the attribute does not exist, it is added to the window's presentation parameters, with the specified value. (See also [WinQueryPresParam](#) and [WinRemovePresParam](#)).

When a presentation parameter is set, a [WM_PRESPARAMCHANGED](#) message is sent to all windows owned by the window calling the WinSetPresParam function.

When switching from using color indexes to using RGB color values, you must call GpiCreateLogColorTable with the LCOLF_RGB parameter. Otherwise, this function will treat the RGB color passed as an index into a color table, rather than as an absolute color value.

WinSetPresParam - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinSetPresParam - Related Functions

Related Functions

- [WinDrawBitmap](#)
 - [WinDrawBorder](#)
 - [WinDrawPointer](#)
 - [WinDrawText](#)
 - [WinFillRect](#)
 - [WinGetSysBitmap](#)
 - [WinInvertRect](#)
 - [WinQueryPresParam](#)
 - [WinRemovePresParam](#)
 - [WinScrollWindow](#)
 - [WinSetPresParam](#)
-

WinSetPresParam - Example Code

This example changes the border color to blue.

```
#define INCL_WINSYS
#define INCL_GPIBITMAPS /* for RGB2 structure definition. */
#include <OS2.H>

HWND hwnd;
RGB2 rgb2; /* Red, green, and blue color index. */
rgb2.bBlue = 200;
rgb2.bGreen = 10;
rgb2.bRed = 5;
rgb2.fcOptions = 0;

WinSetPresParam(hwnd,
                 PP_BORDERCOLOR,
                 (ULONG)sizeof(RGB2),
                 (PVOID)&rgb2);
```

This example changes the font to 18-point Courier. Note that the length parameter includes 10 characters for the font name and 1 for a null terminator.

```
#define INCL_WINSYS
#define INCL_GPIBITMAPS /* for RGB structure definition
#include <OS2.H>

HWND hwnd;

WinSetPresParam(hwnd,
                 PP_FONTNAME,
                 11,
                 (PVOID)"18.Courier");
```

WinSetPresParam - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSetRect

WinSetRect - Syntax

This function sets rectangle coordinates.

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
PRECTL   prclrect;     /* Rectangle to be updated. */
LONG     lLeft;        /* Left edge of rectangle. */
LONG     lBottom;      /* Bottom edge of rectangle. */
LONG     lRight;       /* Right edge of rectangle. */
LONG     lTop;         /* Top edge of rectangle. */
BOOL     rc;           /* Success indicator. */

rc = WinSetRect(hab, prclrect, lLeft, lBottom,
               lRight, lTop);
```

WinSetRect Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinSetRect Parameter - prclrect

prclrect ([PRECTL](#)) - in/out

Rectangle to be updated.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type **WRECT** may also be used, if supported by the language.

WinSetRect Parameter - ILeft

ILeft (**LONG**) - input
Left edge of rectangle.

WinSetRect Parameter - IBottom

IBottom (**LONG**) - input
Bottom edge of rectangle.

WinSetRect Parameter - IRight

IRight (**LONG**) - input
Right edge of rectangle.

WinSetRect Parameter - ITop

ITop (**LONG**) - input
Top edge of rectangle.

WinSetRect Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetRect - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

prclrect ([PRECTL](#)) - in/out
Rectangle to be updated.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) may also be used, if supported by the language.

lLeft ([LONG](#)) - input
Left edge of rectangle.

lBottom ([LONG](#)) - input
Bottom edge of rectangle.

lRight ([LONG](#)) - input
Right edge of rectangle.

lTop ([LONG](#)) - input
Top edge of rectangle.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetRect - Remarks

This function is equivalent to assigning the left, top, right, and bottom arguments to the appropriate fields of [RECTL](#).

WinSetRect - Related Functions

Related Functions

- [WinCopyRect](#)
 - [WinEqualRect](#)
 - [WinFillRect](#)
 - [WinInflateRect](#)
 - [WinIntersectRect](#)
 - [WinIsRectEmpty](#)
 - [WinOffsetRect](#)
 - [WinPtInRect](#)
 - [WinSetRect](#)
 - [WinSetRectEmpty](#)
 - [WinSubtractRect](#)
 - [WinUnionRect](#)
-

WinSetRect - Example Code

This example calls WinQueryWindowRect to get the dimensions of the window, and then calls WinSetRect to downsize it.

```
#define INCL_WINRECTANGLES
#include <OS2.H>
HAB hab;
RECTL rcl;
HWND hwnd;

WinQueryWindowRect(hwnd, &rcl);          /* get window dimensions */
WinSetRect(hab,&rcl,
           rcl.xLeft - 10,
           rcl.yBottom -10,
           rcl.xRight - 10,
           rcl.yTop - 10);
```

WinSetRect - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSetRectEmpty

WinSetRectEmpty - Syntax

This function sets a rectangle empty.

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
PRECTL   prclrect;     /* Rectangle to be set empty. */
BOOL     rc;           /* Success indicator. */

rc = WinSetRectEmpty(hab, prclrect);
```

WinSetRectEmpty Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinSetRectEmpty Parameter - prclrect

prclrect ([PRECTL](#)) - in/out
Rectangle to be set empty.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) may also be used, if supported by the language.

WinSetRectEmpty Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetRectEmpty - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

prclrect ([PRECTL](#)) - in/out
Rectangle to be set empty.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) may also be used, if supported by the language.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetRectEmpty - Remarks

This function is equivalent to a [WinSetRect](#) (*hab*, *prclrect*, 0, 0, 0, 0) call.

WinSetRectEmpty - Related Functions

Related Functions

- [WinCopyRect](#)
- [WinEqualRect](#)
- [WinFillRect](#)
- [WinInflateRect](#)
- [WinIntersectRect](#)
- [WinIsRectEmpty](#)
- [WinOffsetRect](#)
- [WinPtInRect](#)
- [WinSetRect](#)
- [WinSetRectEmpty](#)
- [WinSubtractRect](#)
- [WinUnionRect](#)

WinSetRectEmpty - Example Code

This example calls WinSetRectEmpty to empty the rectangle structure.

```
#define INCL_WINRECTANGLES
#include <OS2.H>
HAB hab;
RECTL rcl;

WinSetRectEmpty(hab,&rcl);
```

WinSetRectEmpty - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Example Code](#)
 - [Related Functions](#)
 - [Glossary](#)

WinSetSynchroMode

WinSetSynchroMode - Syntax

This function is intended for use in a distributed application.

```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HAB      hab;      /* Anchor-block handle. */
LONG     lMode;     /* Synchronization mode. */
BOOL     rc;        /* Success indicator. */

rc = WinSetSynchroMode(hab, lMode);
```

WinSetSynchroMode Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinSetSynchroMode Parameter - lMode

lMode (**LONG**) - input
Synchronization mode.

SSM_SYNCHRONOUS	Synchronous mode
SSM_ASYNCHRONOUS	Asynchronous mode
SSM_MIXED	Mixed mode.

WinSetSynchroMode Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetSynchroMode - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

IMode ([LONG](#)) - input
Synchronization mode.

SSM_SYNCHRONOUS
Synchronous mode
SSM_ASYNCHRONOUS
Asynchronous mode
SSM_MIXED
Mixed mode.

rc ([BOOL](#)) - returns
Success indicator.

TRUE
Successful completion
FALSE
Error occurred.

WinSetSynchroMode - Remarks

This function allows an application whose message queue is distributed, to synchronize the processing of those messages. This is achieved by the use of the [MsgControlHook](#) hook which is invoked by this function.

WinSetSynchroMode - Related Functions

Related Functions

- [WinBroadcastMsg](#)
 - [WinCreateMsgQueue](#)
 - [WinDestroyMsgQueue](#)
 - [WinDispatchMsg](#)
 - [WinGetDlgMsg](#)
 - [WinGetMsg](#)
 - [WinInSendMessage](#)
 - [WinPeekMsg](#)
 - [WinPostMsg](#)
 - [WinPostQueueMsg](#)
 - [WinQueryMsgPos](#)
 - [WinQueryMsgTime](#)
 - [WinQueryQueueInfo](#)
 - [WinQueryQueueStatus](#)
 - [WinSendDlgItemMsg](#)
 - [WinSendMessage](#)
 - [WinSetClassMsgInterest](#)
 - [WinSetMsgInterest](#)
 - [WinSetMsgMode](#)
 - [WinSetSynchroMode](#)
 - [WinWaitMsg](#)
-

WinSetSynchroMode - Example Code

This function is intended for use in an application with a distributed queue.

```
#define INCL_WINMESSAGEGR
#include <OS2.H>
HAB hab;
WinSetSynchroMode(hab,
                  SSM_SYNCHRONOUS); /* synchronous mode. */
```

WinSetSynchroMode - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinSetSysColors

WinSetSysColors - Syntax

This function sets system color values.

```
#define INCL_WINSYS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND hwndDesktop; /* Desktop-window handle. */
ULONG flOptions; /* Options. */
ULONG ulFormat; /* Format of entries in the table, as follows. */
LONG lStart; /* Starting system color index. */
ULONG ulTablen; /* Number of elements. */
PLONG alTable; /* Table. */
BOOL rc; /* Success indicator. */

rc = WinSetSysColors(hwndDesktop, flOptions,
                    ulFormat, lStart, ulTablen, alTable);
```

WinSetSysColors Parameter - hwndDesktop

hwndDesktop ([HWND](#)) - input

Desktop-window handle.

HWND_DESKTOP

The desktop-window handle

Other

Specified desktop-window handle.

WinSetSysColors Parameter - flOptions

flOptions ([ULONG](#)) - input
Options.

LCOL_RESET

The system colors are all to be reset to default before processing the remainder of the data in this function.

LCOL_PURECOLOR

Color-dithering should not be used to create colors not available in the physical palette. If this option is set, only pure colors are used and no dithering is done.

WinSetSysColors Parameter - ulFormat

ulFormat ([ULONG](#)) - input
Format of entries in the table, as follows.

LCOLF_INDRGB

Array of (index,RGB) values. Each pair of entries is 8-bytes long, comprising 4 bytes for the index, and 4 bytes for the color value. For system color indexes, see *lStart*.

LCOLF_CONSECRGB

Array of (RGB) values, corresponding to color indexes *lStart* upwards. Each entry is 4-bytes long.

WinSetSysColors Parameter - lStart

lStart ([LONG](#)) - input
Starting system color index.

This parameter is applicable only if the *ulFormat* parameter is set to LCOLF_CONSECRGB.

The number of system colors (as defined below) is given by SYSCLR_CSYS_COLORS.

The following system color indexes are defined (each successive index is one larger than its predecessor):

SYSCLR_ENTRYFIELD

Entry field and list box background color.

SYSCLR_MENUDISABLEDTEXT

Entry field background color.

SYSCLR_MENUHILITE

Selected menu item text.

`SYSLR_MENUHILITEBGND`
Selected menu item background.

`SYSLR_PAGEBACKGROUND`
Notebook page background.

`SYSLR_FIELDBACKGROUND`
Inactive scroll bar and default control background color.

`SYSLR_BUTTONLIGHT`
Light push button (3D effect).

`SYSLR_BUTTONMIDDLE`
Middle push button (3D effect).

`SYSLR_BUTTONDARK`
Dark push button (3D effect).

`SYSLR_BUTTONDEFAULT`
Push button.

`SYSLR_TITLEBOTTOM`
Line drawn under title bar.

`SYSLR_SHADOW`
Drop shadow for menus and dialogs.

`SYSLR_ICONTEXT`
Text written under icons on the desktop.

`SYSLR_DIALOGBACKGROUND`
Pop up dialog box background.

`SYSLR_HILITEFOREGROUND`
Selection foreground.

`SYSLR_HILITEBACKGROUND`
Selection background.

`SYSLR_INACTIVETITLETEXTBKGD`
Background of inactive title text.

`SYSLR_ACTIVETITLETEXTBKGD`
Background of active title text.

`SYSLR_INACTIVETITLETEXT`
Inactive title text.

`SYSLR_ACTIVETITLETEXT`
Active title text.

`SYSLR_OUTPUTTEXT`
Output text.

`SYSLR_WINDOWSTATICTEXT`
Static (nonselectable) text.

`SYSLR_SCROLLBAR`
Active scroll bar background area.

`SYSLR_BACKGROUND`
Desktop background.

`SYSLR_ACTIVETITLE`
Active window title.

`SYSLR_INACTIVETITLE`
Inactive window title.

`SYSLR_MENU`
Menu background.

`SYSLR_WINDOW`

	Window background.
SYSCLR_WINDOWFRAME	Window frame (border line).
SYSCLR_MENUTEXT	Normal menu item text.
SYSCLR_WINDOWTEXT	Window text.
SYSCLR_TITLETEXT	Text in title bar, size box, scroll bar arrow box.
SYSCLR_ACTIVEBORDER	Border fill of active window.
SYSCLR_INACTIVEBORDER	Border fill of inactive window.
SYSCLR_APPWORKSPACE	Background of specific main windows.
SYSCLR_HELPBACKGROUND	Background of help panels.
SYSCLR_HELPTEXT	Help text.
SYSCLR_HELPHILITE	Highlighted help text.
SYSCLR_SHADOWHILITEBGND	Shadows of workplace object background highlight color.
SYSCLR_SHADOWHILITEFGND	Shadows of workplace object foreground highlight color.
SYSCLR_SHADOWTEXT	Shadows of workplace object text color.

WinSetSysColors Parameter - ulTablen

ulTablen ([ULONG](#)) - input
Number of elements.

Number of elements supplied in *aITable*. This may be 0 if, for example, the color table is merely to be reset to the default. For LCOLF_INDRGB it must be an even number.

WinSetSysColors Parameter - aITable

aITable ([PLONG](#)) - input
Table.

Start address of the application data area, containing the color-table definition data. The format depends on the value of *ulFormat*.

Each color value is a 4-byte integer, with a value of

$(R * 65536) + (G * 256) + B$

where:

R is red intensity value
G is green intensity value
B is blue intensity value.

There are 8 bits for each primary; the maximum intensity for each primary is 255.

WinSetSysColors Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE Successful completion
FALSE Error occurred.

WinSetSysColors - Parameters

hWndDesktop (**HWND**) - input
Desktop-window handle.

HWND_DESKTOP The desktop-window handle
Other Specified desktop-window handle.

flOptions (**ULONG**) - input
Options.

LCOL_RESET The system colors are all to be reset to default before processing the remainder of the data in this function.

LCOL_PURECOLOR Color-dithering should not be used to create colors not available in the physical palette. If this option is set, only pure colors are used and no dithering is done.

uiFormat (**ULONG**) - input
Format of entries in the table, as follows.

LCOLF_INDRGB Array of (index,RGB) values. Each pair of entries is 8-bytes long, comprising 4 bytes for the index, and 4 bytes for the color value. For system color indexes, see */Start*.

LCOLF_CONSECRGB Array of (RGB) values, corresponding to color indexes */Start* upwards. Each entry is 4-bytes long.

iStart (**LONG**) - input
Starting system color index.

This parameter is applicable only if the *uiFormat* parameter is set to LCOLF_CONSECRGB.

The number of system colors (as defined below) is given by SYSCLR_CSYS_COLORS.

The following system color indexes are defined (each successive index is one larger than its predecessor):

SYSCLR_ENTRYFIELD
Entry field and list box background color.

SYSCLR_MENUDISABLEDTEXT
Entry field background color.

SYSCLR_MENUHILITE
Selected menu item text.

SYSCLR_MENUHILITEBGND
Selected menu item background.

SYSCLR_PAGEBACKGROUND
Notebook page background.

SYSCLR_FIELDBACKGROUND
Inactive scroll bar and default control background color.

SYSCLR_BUTTONLIGHT
Light push button (3D effect).

SYSCLR_BUTTONMIDDLE
Middle push button (3D effect).

SYSCLR_BUTTONDARK
Dark push button (3D effect).

SYSCLR_BUTTONDEFAULT
Push button.

SYSCLR_TITLEBOTTOM
Line drawn under title bar.

SYSCLR_SHADOW
Drop shadow for menus and dialogs.

SYSCLR_ICONTEXT
Text written under icons on the desktop.

SYSCLR_DIALOGBACKGROUND
Pop up dialog box background.

SYSCLR_HILITEFOREGROUND
Selection foreground.

SYSCLR_HILITEBACKGROUND
Selection background.

SYSCLR_INACTIVETITLETEXTBKGD
Background of inactive title text.

SYSCLR_ACTIVETITLETEXTBKGD
Background of active title text.

SYSCLR_INACTIVETITLETEXT
Inactive title text.

SYSCLR_ACTIVETITLETEXT
Active title text.

SYSCLR_OUTPUTTEXT
Output text.

SYSCLR_WINDOWSTATICTEXT
Static (nonselectable) text.

SYSCLR_SCROLLBAR
Active scroll bar background area.

SYSLR_BACKGROUND
 Desktop background.

SYSLR_ACTIVETITLE
 Active window title.

SYSLR_INACTIVETITLE
 Inactive window title.

SYSLR_MENU
 Menu background.

SYSLR_WINDOW
 Window background.

SYSLR_WINDOWFRAME
 Window frame (border line).

SYSLR_MENUTEXT
 Normal menu item text.

SYSLR_WINDOWTEXT
 Window text.

SYSLR_TITLETEXT
 Text in title bar, size box, scroll bar arrow box.

SYSLR_ACTIVEBORDER
 Border fill of active window.

SYSLR_INACTIVEBORDER
 Border fill of inactive window.

SYSLR_APPWORKSPACE
 Background of specific main windows.

SYSLR_HELPBACKGROUND
 Background of help panels.

SYSLR_HELPTEXT
 Help text.

SYSLR_HELPHILITE
 Highlighted help text.

SYSLR_SHADOWHILITEBGND
 Shadows of workplace object background highlight color.

SYSLR_SHADOWHILITEFGND
 Shadows of workplace object foreground highlight color.

SYSLR_SHADOWTEXT
 Shadows of workplace object text color.

uiTablen (ULONG) - input
 Number of elements.

Number of elements supplied in *aTable*. This may be 0 if, for example, the color table is merely to be reset to the default. For LCOLF_INDRGB it must be an even number.

aTable (PLONG) - input
 Table.

Start address of the application data area, containing the color-table definition data. The format depends on the value of *uiFormat*.

Each color value is a 4-byte integer, with a value of

$$(R * 65536) + (G * 256) + B$$

where:

R is red intensity value
G is green intensity value
B is blue intensity value.

There are 8 bits for each primary; the maximum intensity for each primary is 255.

rc (**BOOL**) - returns
Success indicator.

TRUE Successful completion
FALSE Error occurred.

WinSetSysColors - Remarks

This function sends all main windows in the system a [WM_SYSCOLORCHANGE](#) message to indicate that the colors have changed. When this message is received, applications that depend on the system colors can query the new color values with the [WinQuerySysColor](#) function.

After the [WM_SYSCOLORCHANGE](#) messages are sent, all windows in the system are invalidated so that they are redrawn with the new system colors.

This function does *not* write any system color changes to the initialization file See

The following table gives the default RGB values for each color index:

System Color Index	Default Color	Default RGB Values
SYSCLR_ACTIVEBORDER	Pale yellow	255 255 128
SYSCLR_ACTIVETITLE	Teal	64 128 128
SYSCLR_ACTIVETITLETEXT	White	255 255 255
SYSCLR_ACTIVETITLETEXTBGND	Teal	64 128 128
SYSCLR_APPWORKSPACE	Off-white	255 255 224
SYSCLR_BACKGROUND	Light gray	204 204 204
SYSCLR_BUTTONDARK	Dark gray	128 128 128
SYSCLR_BUTTONDEFAULT	Black	0 0 0
SYSCLR_BUTTONLIGHT	White	255 255 255
SYSCLR_BUTTONMIDDLE	Light gray	204 204 204
SYSCLR_DIALOGBACKGROUND	Light gray	204 204 204
SYSCLR_ENTRYFIELD	Pale yellow	255 255 204
SYSCLR_FIELDBACKGROUND	Light gray	204 204 204
SYSCLR_HELPBACKGROUND	White	255 255 255
SYSCLR_HELPHILITE	Blue green	0 128 128
SYSCLR_HELPTEXT	Dark blue	0 0 128
SYSCLR_HILITEBACKGROUND	Dark gray	128 128 128
SYSCLR_HILITEFOREGROUND	White	255 255 255
SYSCLR_ICONTEXT	Black	0 0 0

SYSCLR_INACTIVEBORDER	Light gray	204	204	204
SYSCLR_INACTIVETITLE	Light gray	204	204	204
SYSCLR_INACTIVETITLETEXT	Dark gray	128	128	128
SYSCLR_INACTIVETITLETEXTBGND	Light gray	204	204	204
SYSCLR_MENU	Light gray	204	204	204
SYSCLR_MENUDISABLEDTEXT	Dark gray	128	128	128
SYSCLR_MENUHILITE	Black	0	0	0
SYSCLR_MENUHILITEBGND	Light grey	204	204	204
SYSCLR_MENUTEXT	Black	0	0	0
SYSCLR_OUTPUTTEXT	Black	0	0	0
SYSCLR_PAGEBACKGROUND	White	255	255	255
SYSCLR_SCROLLBAR	Pale gray	192	192	192
SYSCLR_SHADOW	Dark gray	128	128	128
SYSCLR_SHADOWHILITEBGND	Dark gray	128	128	128
SYSCLR_SHADOWHILITEFGND	White	255	255	255
SYSCLR_SHADOWTEXT	Dark gray	128	128	128
SYSCLR_TITLEBOTTOM	Dark gray	128	128	128
SYSCLR_TITLETEXT	White	255	255	255
SYSCLR_WINDOW	White	255	255	255
SYSCLR_WINDOWFRAME	Dark gray	128	128	128
SYSCLR_WINDOWSTATICTEXT	Blue	0	0	128
SYSCLR_WINDOWTEXT	Black	0	0	0

WinSetSysColors - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_INVALID_FLAG (0x1019)
An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

PMERR_PARAMETER_OUT_OF_RANGE (0x1003)
The value of a parameter was not within the defined valid range for that parameter.

WinSetSysColors - Related Functions

Related Functions

- [WinQuerySysColor](#)

- [WinSetSysColors](#)

WinSetSysColors - Related Messages

Related Messages

- [WM_SYSCOLORCHANGE](#)
-

WinSetSysColors - Example Code

This example changes the desktop background to blue and the output text to green.

```
#define INCL_WINSYS
#define INCL_GPILOGCOLORTABLE
#include <OS2.H>

typedef struct {
    LONG index;
    LONG color;
} ENTRY;

LONG R, G ,B;

ENTRY alTable[2]; /* array of two color/index entries. */

R = 5L; G = 5L; B = 200L;
alTable[0].color = (R * 65536L) + (G * 256L) + B;
R = 5; G = 200; B = 5;
alTable[1].color = (R * 65536L) + (G * 256L) + B;
alTable[0].index = SYSCLR_OUTPUTTEXT; /* output text. */
alTable[1].index = SYSCLR_BACKGROUND; /* desktop background. */

WinSetSysColors (HWND_DESKTOP,
                LCOL_RESET, /* reset system colors before */
                                /* processing remainder of this */
                                /* call. */
                LCOLF_INDRGB, /* Array of (index,RGB) */
                                /* values. Each pair of */
                                /* entries is 8 bytes */
                                /* long, comprising 4 */
                                /* bytes for the index, */
                                /* and 4 bytes for the */
                                /* color value. For */
                                /* system color indexes, */
                                /* see lStart. */
                0L, /* not applicable. */
                (ULONG)4,
                (PLONG)&alTable[0].index);
```

WinSetSysColors - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)

[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinSetSysModalWindow

WinSetSysModalWindow - Syntax

This function makes a window become the system-modal window, or ends the system-modal state.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndDesktop; /* Desktop-window handle, or HWND_DESKTOP. */
HWND    hwnd;         /* Handle of window to become system-modal window. */
BOOL    rc;           /* Success indicator. */

rc = WinSetSysModalWindow(hwndDesktop, hwnd);
```

WinSetSysModalWindow Parameter - hwndDesktop

hwndDesktop ([HWND](#)) - input
Desktop-window handle, or HWND_DESKTOP.

WinSetSysModalWindow Parameter - hwnd

hwnd ([HWND](#)) - input
Handle of window to become system-modal window.

If NULLHANDLE, system-modal state is ended, and input processing returns to its normal state.

WinSetSysModalWindow Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetSysModalWindow - Parameters

hwndDesktop ([HWND](#)) - input
Desktop-window handle, or HWND_DESKTOP.

hwnd ([HWND](#)) - input
Handle of window to become system-modal window.

If NULLHANDLE, system-modal state is ended, and input processing returns to its normal state.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetSysModalWindow - Remarks

Input processing can enter a "system modal" state. In this state, all pointing device and keyboard input is directed to a special window, known as the system-modal window, or to one of its child windows (or a window owned by one of them). An "owned" window is a window that refers to its owner window set by using either the *hwndOwner* parameter of the [WinCreateWindow](#) function or the *hwndNewOwner* parameter of the [WinSetOwner](#) function. All other main windows behave as though they are disabled and no interaction is possible with them.

Note: The disabled windows are not actually disabled, but made noninteractive. No messages are sent to these windows when the system-modal state is entered or left, and their WS_DISABLE style bits are not changed.

Where a system-modal window exists and another window is explicitly made the active window, the newly activated window becomes the system-modal window. This replaces the old one, which becomes a noninteractive window. When the system-modal window is destroyed, the system-modal state is ended, and input processing returns to its normal state.

This function should only be called while processing pointing device or keyboard input.

The new system-modal window is **not** locked during the processing of this function.

WinSetSysModalWindow - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinSetSysModalWindow - Related Functions

Related Functions

- [WinQuerySysModalWindow](#)
- [WinSetSysModalWindow](#)

WinSetSysModalWindow - Example Code

This example uses the WinSetModalWindow to set a system modal window.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
HWND hwndSysModal;

/* Input processing can enter a "system modal" state. In */
/* this state, all pointing device and keyboard input */
/* is directed to a special window, known as the */
/* system-modal window. Typically, this will be a dialog */
/* window requiring input. */

WinSetSysModalWindow(HWND_DESKTOP,hwndSysModal);
```

WinSetSysModalWindow - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSetSysPointerData

WinSetSysPointerData - Syntax

This function is specific to OS/2 Version 2.1 or higher. This function sets the given system pointer to the new icon specified by the [ICONINFO](#) structure.

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND          hwndDesktop; /* Handle to the desktop window. */
```



```
ULONG      iptr;          /* Index of the desired system pointer. */
PICONINFO  pIconInfo;     /* Icon data. */
BOOL       rc;            /* Return value. */

rc = WinSetSysPointerData(hwndDesktop, iptr,
                          pIconInfo);
```

WinSetSysPointerData Parameter - hwndDesktop

hwndDesktop ([HWND](#)) - input
Handle to the desktop window.

WinSetSysPointerData Parameter - iptr

iptr ([ULONG](#)) - input
Index of the desired system pointer.

WinSetSysPointerData Parameter - pIconInfo

pIconInfo ([PICONINFO](#)) - in/out
Icon data.

New icon data for the requested system pointer or NULL to reset the system pointer to its default appearance.

WinSetSysPointerData Return Value - rc

rc ([BOOL](#)) - returns
Return value.

TRUE	Successful.
FALSE	An error occurred.

WinSetSysPointerData - Parameters

hwndDesktop ([HWND](#)) - input
Handle to the desktop window.

iptr ([ULONG](#)) - input
Index of the desired system pointer.

plconInfo ([PICONINFO](#)) - in/out
Icon data.

New icon data for the requested system pointer or NULL to reset the system pointer to its default appearance.

rc ([BOOL](#)) - returns
Return value.

TRUE	Successful.
FALSE	An error occurred.

WinSetSysPointerData - Remarks

This function sets the given system pointer to the new icon specified by the [ICONINFO](#) structure that is passed in. Provided that the icon is valid, the screen will be refreshed with the updated system pointer if necessary.

If NULL is passed for the *plconInfo* parameter, the operating system reverts back to the default pointer shapes defined by the system. All alterations made using WinSetSysPointerData are persistent. They are preserved across IPLs of the system.

WinSetSysPointerData - Related Functions

Related Functions

- [WinCreatePointer](#)
- [WinCreatePointerIndirect](#)
- [WinDestroyPointer](#)
- [WinDrawPointer](#)
- [WinLoadPointer](#)
- [WinQueryPointer](#)
- [WinQueryPointerInfo](#)
- [WinQueryPointerPos](#)
- [WinQuerySysPointer](#)
- [WinQuerySysPointerData](#)
- [WinSetPointer](#)
- [WinSetPointerPos](#)
- [WinSetSysPointerData](#)
- [WinShowPointer](#)

WinSetSysPointerData - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

WinSetSysValue

WinSetSysValue - Syntax

This function sets a system value.

```
#define INCL_WINSYS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndDesktop; /* Desktop-window handle. */
LONG    iSysValue;    /* System-value identity. */
LONG    lValue;       /* System value. */
BOOL    rc;           /* Value-set indicator. */

rc = WinSetSysValue(hwndDesktop, iSysValue,
                    lValue);
```

WinSetSysValue Parameter - hwndDesktop

hwndDesktop ([HWND](#)) - input
Desktop-window handle.

HWND_DESKTOP	Set the system values for the desktop-window handle
Other	Set the system values for the specified desktop-window handle.

WinSetSysValue Parameter - iSysValue

iSysValue ([LONG](#)) - input
System-value identity.

This must be one of the following `SV_*` constants that is marked with an asterisk (*).

Note: Not all system values can be set; those that can be set are marked with an asterisk (*). All of the `SV_*` constants can be queried using the [WinQuerySysValue](#).

SV_ALARM	(*) TRUE if the alarm sound generated by WinAlarm is enabled; FALSE if the alarm sound is disabled.
----------	---

SV_ALTMNEMONIC	(*) TRUE if the mnemonic is made up of KATAKANA characters; FALSE if the mnemonic is made up of ROMAN
----------------	---

characters.

SV_ANIMATION

(*) TRUE when animation is set on. FALSE when animation is set off.

SV_BEGINDRAG

(*) Mouse begin drag (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_BEGINDRAGKB

(*) Keyboard begin drag (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_BEGINSELECT

(*) Mouse begin swipe select (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_BEGINSELECTKB

(*) Keyboard begin swipe select (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_CICONTEXTLINES

(*) Maximum number of lines that the icon text may occupy for a minimized window.

SV_CONTEXTHELP

(*) Mouse control for pop-up menu (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_CONTEXTHELPKB

(*) Keyboard control for pop-up menu (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_CONTEXTMENU

(*) Mouse request pop-up menu (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_CONTEXTMENUKB

(*) Keyboard request pop-up menu (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_CMOUSEBUTTONS

The number of buttons on the pointing device (zero if no pointing device is installed).

SV_CTIMERS

Count of available timers.

SV_CURSORLEVEL

The cursor hide level.

SV_CURSORRATE

(*) Cursor blink rate, in milliseconds.

SV_CXBORDER

Width of the nominal-width border.

SV_CXBYTEALIGN

Horizontal count of pels for alignment.

SV_CXDBLCLK

(*) Width of the pointing device double-click sensitive area. The default is the system-font character width.

SV_CXDLGFRAME

Width of the dialog-frame border.

SV_CXFULLSCREEN

Width of the client area when the window is full screen.

SV_CXHSCROLLARROW

Width of the horizontal scroll-bar arrow bit maps.

SV_CXHSLIDER

Width of the horizontal scroll-bar thumb.

SV_CXICON

Icon width.

SV_CXICONTEXTWIDTH

(*) Maximum number of characters per line allowed in the icon text for a minimized window.

SV_CXMINMAXBUTTON
Width of the minimize/maximize buttons.

SV_CXMOTIONSTART
(*) The number of pels that a pointing device must be moved in the horizontal direction, while the button is depressed, before a WM_BUTTONxMOTIONSTR message is sent.

SV_CXPOINTER
Pointer width.

SV_CXSCREEN
Width of the screen.

SV_CXSIZEBORDER
(*) Width of the sizing border.

SV_CXVSCROLL
Width of the vertical scroll-bar.

SV_CYBORDER
Height of the nominal-width border.

SV_CYBYTEALIGN
Vertical count of pels for alignment.

SV_CYDBLCLK
(*) Height of the pointing device double-click sensitive area. The default is half the height of the system font character height.

SV_CYDLGFRAME
Height of the dialog-frame border.

SV_CYFULLSCREEN
Height of the client area when the window is full screen (excluding menu height).

SV_CYHSCROLL
Height of the horizontal scroll-bar.

SV_CYICON
Icon height.

SV_CYMENU
Height of the single-line menu height.

SV_CYMINMAXBUTTON
Height of the minimize/maximize buttons.

SV_CYMOTIONSTART
(*) The number of pels that a pointing device must be moved in the vertical direction, while the button is depressed, before a WM_BUTTONxMOTIONSTR message is sent.

SV_CYPOINTER
Pointer height.

SV_CYSCREEN
Height of the screen.

SV_CYSIZEBORDER
(*) Height of the sizing border.

SV_CYTITLEBAR
Height of the caption.

SV_CYVSCROLLARROW
Height of the vertical scroll-bar arrow bit maps.

SV_CYVSLIDER
Height of the vertical scroll-bar thumb.

SV_DBLCLKTIME
(*) Pointing device double-click time, in milliseconds.

SV_DEBUG

FALSE indicates this is not a debug system.

SV_ENDDRAG

(*) Mouse end drag (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_ENDDRAGKB

(*) Keyboard end drag (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_ENDSELECT

(*) Mouse select or end swipe select (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_ENDSELECTKB

(*) Keyboard select or end swipe select (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_ERRORDURATION

(*) Duration for error alarms generated by [WinAlarm](#).

SV_ERRORFREQ

(*) Frequency for error alarms generated by [WinAlarm](#).

SV_EXTRAKEYBEEP

(*) When TRUE, the press of a key that does not exist on the Enhanced keyboard causes the system to generate a beep.

SV_FIRSTSCROLLRATE

(*) The delay (in milliseconds) before autoscrolling starts, when using a scroll bar.

SV_INSERTMODE

(*) TRUE if the system is in insert mode (for edit and multi-line edit controls); FALSE if in overwrite mode.

This system value is toggled by the system when the insert key is toggled, regardless of which window has the focus at the time.

SV_KBDALTERED

(*) Hardware ID of the newly attached keyboard.

Note: The OS/2 National Language Support is only loaded once per system IPL. The OS/2 NLS translation is based partially on the type of keyboard device attached to the system. There are two main keyboard device types: PC AT styled and Enhanced styled. Hot Plugging between these two types of devices may result in typing anomalies due to a mismatch in the NLS device tables loaded and that of the attached device. It is strongly recommended that keyboard hot plugging be limited to the device type that the system was IPL'd with. In addition, OS/2 support will default to the 101/102 key Enhanced keyboard if no keyboard or a NetServer Mode password was in use during system IPL. (See Category 4, IOCTls 77h and 7Ah for more information on keyboard devices and types.)

SV_LOCKSTARTINPUT

(*) TRUE when the type ahead function is enabled; FALSE when the type ahead function is disabled.

SV_MENUROLLOLDOWNDELAY

(*) The delay in milliseconds before displaying a pull down referred to from a submenu item, when the button is already down as the pointer moves onto the submenu item.

SV_MENUROLLUPDELAY

(*) The delay in milliseconds before hiding a pull down referred to from a submenu item, when the button is already down as the pointer moves off the submenu item.

SV_MONOICONS

(*) When TRUE preference is given to black and white icons when selecting which icon resource definition to use on the screen. Black and white icons may have more clarity than color icons on LCD and Plasma display screens.

SV_MOUSEPRESENT

When TRUE a mouse pointing device is attached to the system.

SV_NOTEDURATION

(*) Duration for note alarms generated by [WinAlarm](#).

SV_NOTEFREQ

(*) Frequency for note alarms generated by [WinAlarm](#).

SV_OPEN

(*) Mouse open (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_OPENKB
 (*) Keyboard open (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_POINTERLEVEL
 Pointer hide level. If the pointer level is zero, the pointer is visible. If it is greater than zero, the pointer is not visible. The [WinShowPointer](#) call is invoked to increment and decrement the SV_POINTERLEVEL, but its value cannot become negative.

SV_PRINTSCREEN
 (*) TRUE when the Print Screen function is enabled; FALSE when the Print Screen function is disabled.

SV_SCROLLRATE
 (*) The delay (in milliseconds) between scroll operations, when using a scroll bar.

SV_SETLIGHTS
 (*) When TRUE, the appropriate light is set when the keyboard state table is set.

SV_SINGLESELECT
 (*) Mouse select (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_TASKLISTMOUSEACCESS
 (*) Determines whether the task list is displayed when mouse buttons 1 and 2 are pressed simultaneously, or when mouse button 2 is pressed by itself, or for no mouse gesture.

SV_TEXTEDIT
 (*) Mouse begin direct name edit (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_TEXTEDITKB
 (*) Keyboard begin direct name edit (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_TRACKRECTLEVEL
 The hide level of the tracking rectangle (zero if visible, greater than zero if not).

SV_SWAPBUTTON
 (*) TRUE if pointing device buttons are swapped. Normally, the pointing device buttons are set for right-handed use. Setting this value changes them for left-handed use.

If TRUE, WM_LBUTTONDOWN* messages are returned when the user presses the right button, and WM_RBUTTONDOWN* messages are returned when the left button is pressed. Modifying this value affects the entire system. Applications should not normally read or set this value; users update this value by means of the user interface shell to suit their requirements.

SV_WARNINGDURATION
 (*) Duration for warning alarms generated by [WinAlarm](#).

SV_WARNINGFREQ
 (*) Frequency for warning alarms generated by [WinAlarm](#).

WinSetSysValue Parameter - IValue

IValue ([LONG](#)) - input
 System value.

Dimensions are in pels and times are in milliseconds.

WinSetSysValue Return Value - rc

rc (**BOOL**) - returns
Value-set indicator.

TRUE	System value set.
FALSE	Error occurred.

WinSetSysValue - Parameters

hwndDesktop (**HWND**) - input
Desktop-window handle.

HWND_DESKTOP	Set the system values for the desktop-window handle
Other	Set the system values for the specified desktop-window handle.

iSysValue (**LONG**) - input
System-value identity.

This must be one of the following SV_* constants that is marked with an asterisk (*).

Note: Not all system values can be set; those that can be set are marked with an asterisk (*). All of the SV_* constants can be queried using the [WinQuerySysValue](#).

SV_ALARM	(*) TRUE if the alarm sound generated by WinAlarm is enabled; FALSE if the alarm sound is disabled.
SV_ALTMNEMONIC	(*) TRUE if the mnemonic is made up of KATAKANA characters; FALSE if the mnemonic is made up of ROMAN characters.
SV_ANIMATION	(*) TRUE when animation is set on. FALSE when animation is set off.
SV_BEGINDRAG	(*) Mouse begin drag (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).
SV_BEGINDRAGKB	(*) Keyboard begin drag (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).
SV_BEGINSELECT	(*) Mouse begin swipe select (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).
SV_BEGINSELECTKB	(*) Keyboard begin swipe select (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).
SV_CICONTEXTLINES	(*) Maximum number of lines that the icon text may occupy for a minimized window.
SV_CONTEXTHELP	(*) Mouse control for pop-up menu (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).
SV_CONTEXTHELPKB	(*) Keyboard control for pop-up menu (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).
SV_CONTEXTMENU	(*) Mouse request pop-up menu (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).
SV_CONTEXTMENUKB	(*) Keyboard request pop-up menu (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_CMOUSEBUTTONS
The number of buttons on the pointing device (zero if no pointing device is installed).

SV_CTIMERS
Count of available timers.

SV_CURSORLEVEL
The cursor hide level.

SV_CURSRRATE
(*) Cursor blink rate, in milliseconds.

SV_CXBORDER
Width of the nominal-width border.

SV_CXBYTEALIGN
Horizontal count of pels for alignment.

SV_CXDBLCLK
(*) Width of the pointing device double-click sensitive area. The default is the system-font character width.

SV_CXDLGFRAME
Width of the dialog-frame border.

SV_CXFULLSCREEN
Width of the client area when the window is full screen.

SV_CXHSCROLLARROW
Width of the horizontal scroll-bar arrow bit maps.

SV_CXHSLIDER
Width of the horizontal scroll-bar thumb.

SV_CXICON
Icon width.

SV_CXICONTEXTWIDTH
(*) Maximum number of characters per line allowed in the icon text for a minimized window.

SV_CXMINMAXBUTTON
Width of the minimize/maximize buttons.

SV_CXMOTIONSTART
(*) The number of pels that a pointing device must be moved in the horizontal direction, while the button is depressed, before a WM_BUTTONNxmOTIONSTR message is sent.

SV_CXPOINTER
Pointer width.

SV_CXSCREEN
Width of the screen.

SV_CXSIZEBORDER
(*) Width of the sizing border.

SV_CXVSCROLL
Width of the vertical scroll-bar.

SV_CYBORDER
Height of the nominal-width border.

SV_CYBYTEALIGN
Vertical count of pels for alignment.

SV_CYDBLCLK
(*) Height of the pointing device double-click sensitive area. The default is half the height of the system font character height.

SV_CYDLGFRAME
Height of the dialog-frame border.

SV_CYFULLSCREEN
Height of the client area when the window is full screen (excluding menu height).

SV_CYHSCROLL
Height of the horizontal scroll-bar.

SV_CYICON
Icon height.

SV_CYMENU
Height of the single-line menu height.

SV_CYMINMAXBUTTON
Height of the minimize/maximize buttons.

SV_CYMOTIONSTART
(*) The number of pels that a pointing device must be moved in the vertical direction, while the button is depressed, before a WM_BUTTONxMOTIONSTR message is sent.

SV_CYPOINTER
Pointer height.

SV_CYSCREEN
Height of the screen.

SV_CYSIZEBORDER
(*) Height of the sizing border.

SV_CYTITLEBAR
Height of the caption.

SV_CYVSCROLLARROW
Height of the vertical scroll-bar arrow bit maps.

SV_CYVSLIDER
Height of the vertical scroll-bar thumb.

SV_DBLCLKTIME
(*) Pointing device double-click time, in milliseconds.

SV_DEBUG
FALSE indicates this is not a debug system.

SV_ENDDRAG
(*) Mouse end drag (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_ENDDRAGKB
(*) Keyboard end drag (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_ENDSELECT
(*) Mouse select or end swipe select (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_ENDSELECTKB
(*) Keyboard select or end swipe select (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_ERRORDURATION
(*) Duration for error alarms generated by [WinAlarm](#).

SV_ERRORFREQ
(*) Frequency for error alarms generated by [WinAlarm](#).

SV_EXTRAKEYBEEP
(*) When TRUE, the press of a key that does not exist on the Enhanced keyboard causes the system to generate a beep.

SV_FIRSTSCROLLRATE
(*) The delay (in milliseconds) before autoscrolling starts, when using a scroll bar.

SV_INSERTMODE
(*) TRUE if the system is in insert mode (for edit and multi-line edit controls); FALSE if in overwrite mode.

This system value is toggled by the system when the insert key is toggled, regardless of which window has the focus at the time.

SV_KBDALTERED

(*) Hardware ID of the newly attached keyboard.

Note: The OS/2 National Language Support is only loaded once per system IPL. The OS/2 NLS translation is based partially on the type of keyboard device attached to the system. There are two main keyboard device types: PC AT styled and Enhanced styled. Hot Plugging between these two types of devices may result in typing anomalies due to a mismatch in the NLS device tables loaded and that of the attached device. It is strongly recommended that keyboard hot plugging be limited to the device type that the system was IPL'd with. In addition, OS/2 support will default to the 101/102 key Enhanced keyboard if no keyboard or a NetServer Mode password was in use during system IPL. (See Category 4, IOCTls 77h and 7Ah for more information on keyboard devices and types.)

SV_LOCKSTARTINPUT

(*) TRUE when the type ahead function is enabled; FALSE when the type ahead function is disabled.

SV_MENUROLLOLDOWNDELAY

(*) The delay in milliseconds before displaying a pull down referred to from a submenu item, when the button is already down as the pointer moves onto the submenu item.

SV_MENUROLLUPDELAY

(*) The delay in milliseconds before hiding a pull down referred to from a submenu item, when the button is already down as the pointer moves off the submenu item.

SV_MONOICONS

(*) When TRUE preference is given to black and white icons when selecting which icon resource definition to use on the screen. Black and white icons may have more clarity than color icons on LCD and Plasma display screens.

SV_MOUSEPRESENT

When TRUE a mouse pointing device is attached to the system.

SV_NOTEDURATION

(*) Duration for note alarms generated by [WinAlarm](#).

SV_NOTEFREQ

(*) Frequency for note alarms generated by [WinAlarm](#).

SV_OPEN

(*) Mouse open (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_OPENKB

(*) Keyboard open (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_POINTERLEVEL

Pointer hide level. If the pointer level is zero, the pointer is visible. If it is greater than zero, the pointer is not visible. The [WinShowPointer](#) call is invoked to increment and decrement the SV_POINTERLEVEL, but its value cannot become negative.

SV_PRINTSCREEN

(*) TRUE when the Print Screen function is enabled; FALSE when the Print Screen function is disabled.

SV_SCROLLRATE

(*) The delay (in milliseconds) between scroll operations, when using a scroll bar.

SV_SETLIGHTS

(*) When TRUE, the appropriate light is set when the keyboard state table is set.

SV_SINGLESELECT

(*) Mouse select (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_TASKLISTMOUSEACCESS

(*) Determines whether the task list is displayed when mouse buttons 1 and 2 are pressed simultaneously, or when mouse button 2 is pressed by itself, or for no mouse gesture.

SV_TEXTEDIT

(*) Mouse begin direct name edit (low word=mouse message id (WM_*), high word=keyboard control code (KC_*)).

SV_TEXTEDITKB

(*) Keyboard begin direct name edit (low word=virtual key code (VK_*), high word=keyboard control code (KC_*)).

SV_TRACKRECTLEVEL

The hide level of the tracking rectangle (zero if visible, greater than zero if not).

SV_SWAPBUTTON

(*) TRUE if pointing device buttons are swapped. Normally, the pointing device buttons are set for right-handed use. Setting this value changes them for left-handed use.

If TRUE, WM_LBUTTONDOWN* messages are returned when the user presses the right button, and WM_RBUTTONDOWN* messages are returned when the left button is pressed. Modifying this value affects the entire system. Applications should not normally read or set this value; users update this value by means of the user interface shell to suit their requirements.

SV_WARNINGDURATION

(*) Duration for warning alarms generated by [WinAlarm](#).

SV_WARNINGFREQ

(*) Frequency for warning alarms generated by [WinAlarm](#).

IValue ([LONG](#)) - input
System value.

Dimensions are in pels and times are in milliseconds.

rc ([BOOL](#)) - returns
Value-set indicator.

TRUE

System value set.

FALSE

Error occurred.

WinSetSysValue - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_PARAMETER_OUT_OF_RANGE (0x1003)
The value of a parameter was not within the defined valid range for that parameter.

WinSetSysValue - Related Functions

Related Functions

- [WinQuerySysValue](#)
- [WinSetSysValue](#)

WinSetSysValue - Example Code

This example uses the WinSetSysValue call change the sizing border dimensions.

```
#define INCL_WINSYS
#include <OS2.H>
LONG vlXBorder, vlYBorder;

vlXBorder = WinSetSysValue(HWND_DESKTOP,
```

```
        SV_CXSIZEBORDER,  
        20L);  
vlyBorder = WinSetSysValue(HWND_DESKTOP,  
        SV_CYSIZEBORDER,  
        20L);
```

WinSetSysValue - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinSetVisibleRegionNotify

WinSetVisibleRegionNotify - Syntax

This function allows an application to request that a given window receive notifications every time that its visible region gets altered.

```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, */  
#include <os2.h>  
  
HWND    hwnd;        /* The window handle. */  
BOOL    fEnable;     /* Enable flag. */  
BOOL    fSuccess;     /* Success indicator. */  
  
fSuccess = WinSetVisibleRegionNotify(hwnd,  
        fEnable);
```

WinSetVisibleRegionNotify Parameter - hwnd

hwnd ([HWND](#)) - input
The window handle.

WinSetVisibleRegionNotify Parameter - fEnable

fEnable (BOOL) - input
Enable flag.

TRUE

The specified window wants to receive notifications every time its visible region is modified.

FALSE

The specified window no longer wants to receive any notification messages concerning its visible region.

WinSetVisibleRegionNotify Return Value - fSuccess

fSuccess (BOOL) - returns
Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinSetVisibleRegionNotify - Parameters

hwnd (HWND) - input
The window handle.

fEnable (BOOL) - input
Enable flag.

TRUE

The specified window wants to receive notifications every time its visible region is modified.

FALSE

The specified window no longer wants to receive any notification messages concerning its visible region.

fSuccess (BOOL) - returns
Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinSetVisibleRegionNotify - Remarks

This call causes a notification message to be sent to the specified window every time its visible area is modified on the screen. In addition, the window is notified when an application issues a [WinLockWindowUpdate](#) call that affects it, so that any asynchronous drawing activity can be suspended.

Applications that blit to the screen memory directly can use the notification messages [WM_VRNEENABLED](#) and [WM_VRNDISABLED](#) to determine when to suspend blitting and also to determine when the clipping areas for the visible region have been modified.

Use [WinQueryVisibleRegion](#) to obtain the current visible region when a window is notified that its visible region has changed.

WinSetVisibleRegionNotify - Related Functions

Related Functions

- [WinLockWindowUpdate](#)
 - [WinQueryVisibleRegion](#)
-

WinSetVisibleRegionNotify - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

WinSetWindowBits

WinSetWindowBits - Syntax

This function sets a number of bits into the memory of the reserved window words.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND     hwnd; /* Window handle. */
LONG     index; /* Zero-based index of the value to be set. */
ULONG    flData; /* Bit data to store in the window words. */
ULONG    flMask; /* Bits to be written indicator. */
BOOL     rc; /* Success indicator. */

rc = WinSetWindowBits(hwnd, index, flData,
    flMask);
```

WinSetWindowBits Parameter - hwnd

hwnd ([HWND](#)) - input
Window handle.

WinSetWindowBits Parameter - index

index ([LONG](#)) - input
Zero-based index of the value to be set.

The units of *index* are bytes. Valid values are zero through (*cbWindowData* -4), where *cbWindowData* is the parameter in [WinRegisterClass](#) that specifies the number of bytes available for application-defined storage. Any of the following QWL_* values can be specified:

QWL_HMQ	Handle of message queue of window. Note that the leading 16 bits of this value are zero.
QWL_STYLE	Window style.
QWL_HWNDFOCUSSAVE	Window handle of the child windows of this window that last possessed the focus when this frame window was last deactivated.
QWL_USER	<p>A ULONG value for applications to use is present at offset QWL_USER in windows of the following preregistered window classes:</p> <div>WC_FRAME (includes dialog windows) WC_COMBOBOX WC_BUTTON WC_MENU WC_STATIC WC_ENTRYFIELD WC_LISTBOX WC_SCROLLBAR WC_TITTLEBAR WC_MLE WC_SPINBUTTON WC_CONTAINER WC_SLIDER WC_VALUESET WC_NOTEBOOK</div> <p>This value can be used to place application-specific data in controls.</p>
QWL_DEFBUTTON	<p>The default push button for a dialog.</p> <p>The default push button is the one that sends its WM_COMMAND message when the enter key is pressed.</p>
QWL_PENDATA	Reserved for use by operating system extensions. It allows an operating system extension to store data on a per window basis.
Other	Zero-based index.

WinSetWindowBits Parameter - flData

flData (ULONG) - input
Bit data to store in the window words.

This is done under the control of the *flMask* parameter.

WinSetWindowBits Parameter - flMask

flMask (ULONG) - input
Bits to be written indicator.

A "1" bit indicates that the corresponding bit of the *flData* parameter is to be stored into the window word. A "0" bit indicates that the corresponding bit of the *flData* parameter is to be ignored in the storing operation; the value of that bit position in the window word is unaltered.

WinSetWindowBits Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetWindowBits - Parameters

hwnd (HWND) - input
Window handle.

index (LONG) - input
Zero-based index of the value to be set.

The units of *index* are bytes. Valid values are zero through (*cbWindowData* -4), where *cbWindowData* is the parameter in [WinRegisterClass](#) that specifies the number of bytes available for application-defined storage. Any of the following QWL_* values can be specified:

QWL_HMQ	Handle of message queue of window. Note that the leading 16 bits of this value are zero.
QWL_STYLE	Window style.
QWL_HWNDFOCUSSAVE	Window handle of the child windows of this window that last possessed the focus when this frame window was last deactivated.
QWL_USER	A ULONG value for applications to use is present at offset QWL_USER in windows of the following preregistered window classes: WC_FRAME (includes dialog windows)

WC_COMBOBOX
WC_BUTTON
WC_MENU
WC_STATIC
WC_ENTRYFIELD
WC_LISTBOX
WC_SCROLLBAR
WC_TITTLBAR
WC_MLE
WC_SPINBUTTON
WC_CONTAINER
WC_SLIDER
WC_VALUESET
WC_NOTEBOOK

This value can be used to place application-specific data in controls.

QWL_DEFBUTTON

The default push button for a dialog.

The default push button is the one that sends its [WM_COMMAND](#) message when the enter key is pressed.

QWL_PENDATA

Reserved for use by operating system extensions. It allows an operating system extension to store data on a per window basis.

Other

Zero-based index.

flData ([ULONG](#)) - input

Bit data to store in the window words.

This is done under the control of the *flMask* parameter.

flMask ([ULONG](#)) - input

Bits to be written indicator.

A "1" bit indicates that the corresponding bit of the *flData* parameter is to be stored into the window word. A "0" bit indicates that the corresponding bit of the *flData* parameter is to be ignored in the storing operation; the value of that bit position in the window word is unaltered.

rc ([BOOL](#)) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinSetWindowBits - Remarks

The bits are set in a single operation.

WinSetWindowBits - Related Functions

Related Functions

- [WinQueryWindowPtr](#)
- [WinQueryWindowULong](#)
- [WinQueryWindowUShort](#)
- [WinSetWindowBits](#)

- [WinSetWindowPtr](#)
- [WinSetWindowULong](#)
- [WinSetWindowUShort](#)

WinSetWindowBits - Example Code

This example uses the WinSetWindowBits call to change the attributes of a list box so that only one item can be selected. This is done by turning off the multiple-select bit.

```
#define INCL_WINSYS
#include <OS2.H>
HWND hwndMessageLB;
WinSetWindowBits(hwndMessageLB,
    QWL_STYLE,          /* change style bit. */
    0L,                 /* set to 0. */
    LS_MULTIPLESEL);    /* multiple select bit. */
```

WinSetWindowBits - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSetWindowPos

WinSetWindowPos - Syntax

This function allows the general positioning of a window.

Note: Messages may be received from other processes or threads during the processing of this function.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd;          /* Window handle. */
HWND    hwndInsertBehind; /* Relative window-placement order. */
LONG    x;             /* Window position, x-coordinate. */
LONG    y;             /* Window position, y-coordinate. */
LONG    cx;            /* Window size. */
```

```

LONG      cy;                /* Window size. */
ULONG     fl;                /* Window-positioning options. */
BOOL      rc;                /* Repositioning indicator. */

rc = WinSetWindowPos(hwnd, hwndInsertBehind,
    x, y, cx, cy, fl);

```

WinSetWindowPos Parameter - hwnd

hwnd (**HWND**) - input
Window handle.

WinSetWindowPos Parameter - hwndInsertBehind

hwndInsertBehind (**HWND**) - input
Relative window-placement order.

Ignored if SWP_ZORDER is not selected. Values that can be specified are:

HWND_TOP	Place <i>hwnd</i> on top of all siblings
HWND_BOTTOM	Place <i>hwnd</i> behind all siblings
Other	Identifies the sibling window behind which <i>hwnd</i> is to be placed.

WinSetWindowPos Parameter - x

x (**LONG**) - input
Window position, x-coordinate.

This is the x-coordinate of *hwnd*. It is in window coordinates relative to the bottom left corner of its parent, but is ignored if SWP_MOVE is not selected.

WinSetWindowPos Parameter - y

y (**LONG**) - input
Window position, y-coordinate.

This is the y-coordinate of *hwnd*. It is in window coordinates relative to the bottom left corner of its parent, but is ignored if SWP_MOVE is not selected.

WinSetWindowPos Parameter - cx

cx ([LONG](#)) - input
Window size.

This specifies the width of *hwnd* in device units, but is ignored if SWP_SIZE is not selected.

WinSetWindowPos Parameter - cy

cy ([LONG](#)) - input
Window size.

This specifies the depth of *hwnd* in device units, but is ignored if SWP_SIZE is not selected.

WinSetWindowPos Parameter - fl

fl ([ULONG](#)) - input
Window-positioning options.

One or more of these options can be specified:

- SWP_SIZE

Change the window size.
- SWP_MOVE

Change the window x,y position.
- SWP_ZORDER

Change the relative window placement.
- SWP_SHOW

Show the window.
- SWP_HIDE

Hide the window.
- SWP_NOREDRAW

Changes are not redrawn.
- SWP_NOADJUST

Do not send a [WM_ADJUSTWINDOWPOS](#) message before moving or sizing.
- SWP_ACTIVATE

Activate the *hwnd* window if it is a frame window. This indicator has no effect on other windows.

The frame window is made the topmost window, unless SWP_ZORDER is specified also in which instance the *hwndInsertBehind* window is used.
- SWP_DEACTIVATE

Deactivate the *hwnd* window if it is a frame window. This indicator has no effect on other windows.

The frame window is made the bottommost window, unless SWP_ZORDER is specified, in which instance the

hwndInsertBehind window is used.

SWP_MINIMIZE

Minimize the window. This indicator has no effect if the window is in a minimized state, and is also mutually exclusive with SWP_MAXIMIZE and SWP_RESTORE.

SWP_MAXIMIZE

Maximize the window. This indicator has no effect if the window is in a maximized state, and is also mutually exclusive with SWP_MINIMIZE and SWP_RESTORE.

SWP_RESTORE

Restore the window. This indicator has no effect if the window is in its normal state, and is also mutually exclusive with SWP_MINIMIZE and SWP_MAXIMIZE.

The position and size of the window in its normal state is remembered in its window words when it is first maximized or minimized, although these values can be altered by use of the [WinSetWindowUShort](#) function.

The window is restored to the position and size remembered in its window words, unless the SWP_MOVE or SWP_SIZE indicators are set. These indicators cause the position and size values specified in this function to be used.

WinSetWindowPos Return Value - rc

rc ([BOOL](#)) - returns
Repositioning indicator.

TRUE

Window successfully repositioned

FALSE

Window not successfully repositioned.

WinSetWindowPos - Parameters

hwnd ([HWND](#)) - input
Window handle.

hwndInsertBehind ([HWND](#)) - input
Relative window-placement order.

Ignored if SWP_ZORDER is not selected. Values that can be specified are:

HWND_TOP

Place *hwnd* on top of all siblings

HWND_BOTTOM

Place *hwnd* behind all siblings

Other

Identifies the sibling window behind which *hwnd* is to be placed.

x ([LONG](#)) - input
Window position, x-coordinate.

This is the x-coordinate of *hwnd*. It is in window coordinates relative to the bottom left corner of its parent, but is ignored if SWP_MOVE is not selected.

y ([LONG](#)) - input
Window position, y-coordinate.

This is the y-coordinate of *hwnd*. It is in window coordinates relative to the bottom left corner of its parent, but is ignored if

SWP_MOVE is not selected.

cx (**LONG**) - input
Window size.

This specifies the width of *hwnd* in device units, but is ignored if SWP_SIZE is not selected.

cy (**LONG**) - input
Window size.

This specifies the depth of *hwnd* in device units, but is ignored if SWP_SIZE is not selected.

fl (**ULONG**) - input
Window-positioning options.

One or more of these options can be specified:

SWP_SIZE
Change the window size.

SWP_MOVE
Change the window x,y position.

SWP_ZORDER
Change the relative window placement.

SWP_SHOW
Show the window.

SWP_HIDE
Hide the window.

SWP_NOREDRAW
Changes are not redrawn.

SWP_NOADJUST
Do not send a [WM_ADJUSTWINDOWPOS](#) message before moving or sizing.

SWP_ACTIVATE
Activate the *hwnd* window if it is a frame window. This indicator has no effect on other windows.

The frame window is made the topmost window, unless SWP_ZORDER is specified also in which instance the *hwndInsertBehind* window is used.

SWP_DEACTIVATE
Deactivate the *hwnd* window if it is a frame window. This indicator has no effect on other windows.

The frame window is made the bottommost window, unless SWP_ZORDER is specified, in which instance the *hwndInsertBehind* window is used.

SWP_MINIMIZE
Minimize the window. This indicator has no effect if the window is in a minimized state, and is also mutually exclusive with SWP_MAXIMIZE and SWP_RESTORE.

SWP_MAXIMIZE
Maximize the window. This indicator has no effect if the window is in a maximized state, and is also mutually exclusive with SWP_MINIMIZE and SWP_RESTORE.

SWP_RESTORE
Restore the window. This indicator has no effect if the window is in its normal state, and is also mutually exclusive with SWP_MINIMIZE and SWP_MAXIMIZE.

The position and size of the window in its normal state is remembered in its window words when it is first maximized or minimized, although these values can be altered by use of the [WinSetWindowUShort](#) function.

The window is restored to the position and size remembered in its window words, unless the SWP_MOVE or SWP_SIZE indicators are set. These indicators cause the position and size values specified in this function to be used.

rc (**BOOL**) - returns
Repositioning indicator.

TRUE

FALSE	Window successfully repositioned
	Window not successfully repositioned.

WinSetWindowPos - Remarks

If a window created with the CS_SAVEBITS style is reduced, the screen image saved is used to redraw the area uncovered when the window size changes, if those bits are still valid.

If the CS_SIZEREDRAW style is present, the entire window area is assumed invalid if sized. Otherwise, [WM_CALCVALIDRECTS](#) is sent to the window to inform the window manager which bits it may be possible to preserve.

Messages sent from WinSetWindowPos and [WinSetMultWindowPos](#) have specific orderings within the window-positioning process. The process begins with redundancy checks and precalculations on every window for each requested operation. For example, if SWP_SHOW is present but the window is already visible, SWP_SHOW is turned off. If SWP_SIZE is present, and the new size is equal to the old size, SWP_SIZE is turned off.

If the operations create new results, the information is calculated and stored (for instance, when sizing or moving, the new window rectangle is stored for later use). It is at this point that the [WM_ADJUSTWINDOWPOS](#) message is sent to any window that is sizing or moving. It is also at this point that the [WM_CALCVALIDRECTS](#) message is sent to any window that is sizing and does not have the CS_SIZEREDRAW window style.

When all the new window states are calculated, the window-management process begins. Window areas that can be preserved are moved from the old to the new positions, window areas that are invalidated by these operations are calculated and distributed as update regions. When this is finished, and before any synchronous-paint windows are repainted, the [WM_SIZE](#) message is sent to any windows that have changed size. Next, all the synchronous-paint windows that can be are repainted, and the process is complete.

If a synchronous-paint parent window has a size-sensitive area displayed that includes synchronous-paint child windows, the parent needs to reposition those windows when it receives the [WM_SIZE](#) message. Their invalid regions are added to the parent's invalid region, resulting in one update after the parent's [WM_SIZE](#) message, rather than many independent (and later, duplicated) updates.

Note: Some windows will not be positioned precisely to the parameters of this function, but according to the behavior of their window procedure. For example, frame windows without a style creation flag of FCF_NOBYTEALIGN will not position to any specific screen coordinate. Similarly, frame windows with zero size and position are created by the [WinCreateStdWindow](#) function and therefore these values are treated as a special case by the frame window procedure.

Messages sent by this function are:

[WM_ACTIVATE](#)

Sent if a different window becomes the active window. See also [WinSetActiveWindow](#).

[WM_ADJUSTWINDOWPOS](#)

Not sent if SWP_NOADJUST is specified. The message contains an SWP structure that has been filled in by this function with the proposed move/size data. The window can adjust this new position by changing the contents of the [SWP](#) structure. If *hwnd* specifies a frame window, this function recalculates the sizes and positions of the frame controls. If the new window rectangle for any frame control is empty, instead of resizing or repositioning that control, it is hidden if SWP_HIDE is specified. This eliminates needless processing of windows that are not visible. The window rectangle of the control in question is left in its original state. For example, if WinSetWindowPos is issued to change the size of a standard-frame window to an empty rectangle, and [WinQueryWindowRect](#) is issued against the client window, the rectangle returned is not an empty rectangle, but the original client rectangle before WinSetWindowPos was issued.

[WM_CALCVALIDRECTS](#)

Sent to determine the area of a window that may be possible to preserve as the window is sized.

[WM_SIZE](#)

Sent if the size of the window has changed, after the change has been made.

[WM_MOVE](#)

Sent when a window with CS_MOVENOTIFY class style moves its absolute position.

WinSetWindowPos - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_INVALID_FLAG (0x1019)
An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

WinSetWindowPos - Related Functions

Related Functions

- [WinGetMinPosition](#)
- [WinQueryActiveWindow](#)
- [WinQueryWindowPos](#)
- [WinSaveWindowPos](#)
- [WinSetActiveWindow](#)
- [WinSetMultWindowPos](#)
- [WinSetWindowPos](#)

WinSetWindowPos - Related Messages

Related Messages

- [WM_ACTIVATE](#)
- [WM_ADJUSTWINDOWPOS](#)
- [WM_CALCVALIDRECTS](#)
- [WM_ERASEBACKGROUND](#)
- [WM_MOVE](#)
- [WM_SIZE](#)

WinSetWindowPos - Example Code

This example uses the recommended size, position and status from the [WinQueryTaskSize](#) call to position the first window of a newly-started application (typically the main window).

```
#define INCL_WINSWITCHLIST
#define INCL_WINFRAMEMGR
#include <OS2.H>

HAB hab;
SWP winpos;
HWND hwndFrame;

WinQueryTaskSizePos(hab, 0, &winpos);

WinSetWindowPos(hwndFrame, HWND_TOP,
    winpos.x,                /* x pos */
    winpos.y,                /* y pos */
    winpos.cx,               /* x size */
    winpos.cy,               /* y size */
    SWP_ACTIVATE | SWP_MOVE | SWP_SIZE | SWP_SHOW); /* flags*/
```

WinSetWindowPos - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinSetWindowPtr

WinSetWindowPtr - Syntax

This function sets a pointer value into the memory of the reserved window words.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND     hwnd; /* Window handle. */
LONG     lb; /* Zero-based index into the window words. */
PVOID     pp; /* Pointer value to store in the window words. */
BOOL     rc; /* Success indicator. */

rc = WinSetWindowPtr(hwnd, lb, pp);
```

WinSetWindowPtr Parameter - hwnd

hwnd ([HWND](#)) - input
Window handle.

WinSetWindowPtr Parameter - lb

lb (**LONG**) - input
Zero-based index into the window words.

The units of b are bytes. Valid values are zero through (*cbWindowData* -4), where *cbWindowData* is the parameter in [WinRegisterClass](#) that specifies the number of bytes available for application-defined storage.

The value QWP_PFNWP can be used as the index for the address of the window procedure for the window.

WinSetWindowPtr Parameter - pp

pp (**PVOID**) - input
Pointer value to store in the window words.

WinSetWindowPtr Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetWindowPtr - Parameters

hwnd (**HWND**) - input
Window handle.

lb (**LONG**) - input
Zero-based index into the window words.

The units of b are bytes. Valid values are zero through (*cbWindowData* -4), where *cbWindowData* is the parameter in [WinRegisterClass](#) that specifies the number of bytes available for application-defined storage.

The value QWP_PFNWP can be used as the index for the address of the window procedure for the window.

pp (**PVOID**) - input
Pointer value to store in the window words.

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetWindowPtr - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_PARAMETER_OUT_OF_RANGE (0x1003)
The value of a parameter was not within the defined valid range for that parameter.

WinSetWindowPtr - Related Functions

Related Functions

- [WinQueryWindowPtr](#)
 - [WinQueryWindowULong](#)
 - [WinQueryWindowUShort](#)
 - [WinSetWindowBits](#)
 - [WinSetWindowPtr](#)
 - [WinSetWindowULong](#)
 - [WinSetWindowUShort](#)
-

WinSetWindowPtr - Example Code

This function retrieves a pointer value from the memory of the reserved window word.

```
MyWindowProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
    MYINSTANCEDATA *InstanceData; /* application defined structure */

    switch (msg) {
    case WM_CREATE:
        DosAllocMem(&InstanceData, sizeof(MYINSTANCEDATA), fALLOC);
        /* WindowProcedure initializes instance data for this window */
        .
        .
        /* set pointer to instance in window words */
        WinSetWindowPtr(hwnd, 0, InstanceData);
        break;

    case WM_USER + 1: /* application defined message */
        /* Window procedure retrieves instance data to */
        /* process this message */
        InstanceData = WinQueryWindowPtr(hwnd, 0);
        .
        .
        break;
    .
    .
    }
```

WinSetWindowPtr - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSetWindowText

WinSetWindowText - Syntax

This function sets the window text for a specified window.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */  
#include <os2.h>  
  
HWND    hwnd;          /* Window handle. */  
PSZ     pszString;     /* Window text. */  
BOOL    rc;            /* Success indicator. */  
  
rc = WinSetWindowText(hwnd, pszString);
```

WinSetWindowText Parameter - hwnd

hwnd ([HWND](#)) - input
Window handle.

WinSetWindowText Parameter - pszString

pszString ([PSZ](#)) - input
Window text.

WinSetWindowText Return Value - rc

rc ([BOOL](#)) - returns

Success indicator.

TRUE

Text updated

FALSE

Error occurred.

WinSetWindowText - Parameters

hwnd ([HWND](#)) - input
Window handle.

pszString ([PSZ](#)) - input
Window text.

rc ([BOOL](#)) - returns
Success indicator.

TRUE

Text updated

FALSE

Error occurred.

WinSetWindowText - Remarks

This function sends a [WM_SETWINDOWPARAMS](#) message to the window identified by *hwnd*.

If this function references the window of another process, *pszString* must be in memory that is shared by both processes; otherwise, a memory error may occur.

If *hwnd* has a style of `WS_FRAME`, the title-bar window text is set.

Some window classes interpret the *pszString* in a special way. The tilde character (~) indicates that the following character is a mnemonic; for details, see

WinSetWindowText - Errors

Possible returns from [WinGetLastError](#)

`PMERR_INVALID_HWND (0x1001)`
An invalid window handle was specified.

WinSetWindowText - Related Functions

Related Functions

- [WinQueryDlgItemShort](#)
- [WinQueryDlgItemText](#)

- [WinQueryDlgItemTextLength](#)
- [WinQueryWindowText](#)
- [WinQueryWindowTextLength](#)
- [WinSetDlgItemShort](#)
- [WinSetDlgItemText](#)
- [WinSetWindowText](#)

WinSetWindowText - Related Messages

Related Messages

- [WM_SETWINDOWPARAMS](#)

WinSetWindowText - Example Code

This example calls WinQuerySessionTitle to retrieve the application's title, and then sets the title bar of the frame window to that title with WinSetWindowText.

```
#define INCL_WINMESSAGEGR
#define INCL_WINWINDOWMGR
#include <OS2.H>
HAB hab;
HWND hwndFrame, hwndClient;
CHAR szTitle[MAXNAMEL + 1];

WinQuerySessionTitle(hab,
                    0, szTitle,
                    sizeof(szTitle));

hwndFrame = WinQueryWindow(hwndClient,
                          QW_PARENT); /* get handle of parent, */
                                      /* which is frame window. */
WinSetWindowText(hwndFrame, szTitle);
```

WinSetWindowText - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinSetWindowThunkProc

WinSetWindowThunkProc - Syntax

This function associates a pointer-conversion procedure with a window.

```
#define INCL_WINTHUNKAPI /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd;          /* Window handle. */
PFN     pthunkpr;       /* Pointer-conversion procedure identifier. */
BOOL     rc;            /* Success indicator. */

rc = WinSetWindowThunkProc(hwnd, pthunkpr);
```

WinSetWindowThunkProc Parameter - hwnd

hwnd (**HWND**) - input
Window handle.

WinSetWindowThunkProc Parameter - pthunkpr

pthunkpr (**PFN**) - input
Pointer-conversion procedure identifier.

NULL	Any existing pointer-conversion procedure is dissociated from this window.
Other	The pointer-conversion procedure to be associated with this window.

WinSetWindowThunkProc Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	An error occurred.

WinSetWindowThunkProc - Parameters

hwnd (HWND)	- input
Window handle.	
pthunkpr (PFN)	- input
Pointer-conversion procedure identifier.	
NULL	Any existing pointer-conversion procedure is dissociated from this window.
Other	The pointer-conversion procedure to be associated with this window.
rc (BOOL)	- returns
Success indicator.	
TRUE	Successful completion
FALSE	An error occurred.

WinSetWindowThunkProc - Related Functions

- Related Functions**
- [WinQueryClassThunkProc](#)
 - [WinQueryWindowModel](#)
 - [WinQueryWindowThunkProc](#)
 - [WinSetClassThunkProc](#)
 - [WinSetWindowThunkProc](#)

WinSetWindowThunkProc - Example Code

In this example, any thinking procedure is dissociated from the window.

```
#define INCL_WINTHUNKAPI
#include <OS2.H>

HWND hwnd;

WinSetWindowThunkProc(hwnd, NULL);
```

WinSetWindowThunkProc - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Example Code](#)

[Related Functions](#)
[Glossary](#)

WinSetWindowULong

WinSetWindowULong - Syntax

This function sets an unsigned, long integer value into the memory of the reserved window words.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd;    /* Window handle. */
LONG    index;   /* Zero-based index of the value to be set. */
ULONG    ul;     /* Unsigned, long integer value to store in the window words. */
BOOL    rc;      /* Success indicator. */

rc = WinSetWindowULong(hwnd, index, ul);
```

WinSetWindowULong Parameter - hwnd

hwnd ([HWND](#)) - input
Window handle.

WinSetWindowULong Parameter - index

index ([LONG](#)) - input
Zero-based index of the value to be set.

The units of *index* are bytes. Valid values are zero through (*cbWindowData* -4), where *cbWindowData* is the parameter in [WinRegisterClass](#) that specifies the number of bytes available for application-defined storage. Any of the QWL_* values are also valid.

Note: QWS_* values cannot be used.

QWL_DEFBUTTON

The default push button for a dialog.

The default push button is the one that sends its [WM_COMMAND](#) message when the enter key is pressed.

QWL_HMQ

Handle of message queue of window. Note that the leading 16 bits of this value are zero.

QWL_HWNDFOCUSSAVE	Window handle of the child windows of this window that last possessed the focus when this frame window was last deactivated.
QWL_PENDATA	Reserved for use by operating system extensions. It allows an operating system extension to store data on a per window basis.
QWL_STYLE	Window style.
QWL_USER	<p>A ULONG value for applications to use is present at offset QWL_USER in windows of the following preregistered window classes:</p> <div> WC_BUTTON WC_COMBOBOX WC_CONTAINER WC_ENTRYFIELD WC_FRAME (includes dialog windows) WC_LISTBOX WC_MENU WC_MLE WC_NOTEBOOK WC_SCROLLBAR WC_SLIDER WC_SPINBUTTON WC_STATIC WC_TITTLEBAR WC_VALUESET </div> <p>This value can be used to place application-specific data in controls.</p>
Other	Zero-based index.

WinSetWindowULONG Parameter - ul

ul (**ULONG**) - input
 Unsigned, long integer value to store in the window words.

WinSetWindowULONG Return Value - rc

rc (**BOOL**) - returns
 Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetWindowULONG - Parameters

hwnd (**HWND**) - input
Window handle.

index (**LONG**) - input
Zero-based index of the value to be set.

The units of *index* are bytes. Valid values are zero through (*cbWindowData* -4), where *cbWindowData* is the parameter in [WinRegisterClass](#) that specifies the number of bytes available for application-defined storage. Any of the QWL_* values are also valid.

Note: QWS_* values cannot be used.

QWL_DEFBUTTON	The default push button for a dialog. The default push button is the one that sends its WM_COMMAND message when the enter key is pressed.
QWL_HMQ	Handle of message queue of window. Note that the leading 16 bits of this value are zero.
QWL_HWNDFOCUSSAVE	Window handle of the child windows of this window that last possessed the focus when this frame window was last deactivated.
QWL_PENDATA	Reserved for use by operating system extensions. It allows an operating system extension to store data on a per window basis.
QWL_STYLE	Window style.
QWL_USER	A ULONG value for applications to use is present at offset QWL_USER in windows of the following preregistered window classes: WC_BUTTON WC_COMBOBOX WC_CONTAINER WC_ENTRYFIELD WC_FRAME (includes dialog windows) WC_LISTBOX WC_MENU WC_MLE WC_NOTEBOOK WC_SCROLLBAR WC_SLIDER WC_SPINBUTTON WC_STATIC WC_TITTLEBAR WC_VALUESET This value can be used to place application-specific data in controls.
Other	Zero-based index.

ul (**ULONG**) - input
Unsigned, long integer value to store in the window words.

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetWindowULong - Remarks

The specified *index* is valid only if all of the bytes referenced are within the reserved memory.

WinSetWindowULong - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_PARAMETER_OUT_OF_RANGE (0x1003)
The value of a parameter was not within the defined valid range for that parameter.

WinSetWindowULong - Related Functions

Related Functions

- [WinQueryWindowPtr](#)
 - [WinQueryWindowULong](#)
 - [WinQueryWindowUShort](#)
 - [WinSetWindowBits](#)
 - [WinSetWindowPtr](#)
 - [WinSetWindowULong](#)
 - [WinSetWindowUShort](#)
-

WinSetWindowULong - Example Code

This example transfers a pointer from the application-defined data area of a dialog window to the application-defined data area (window word) of a main window. The pointer is then retrieved.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HWND hwndClient;
ULONG msg;
MPARAM pParm, mp1, mp2;

/* Inside dialog procedure */
switch( msg )
{
    case WM_INITDLG:
        /* This points to the data area and is passed by */
        /* WinLoadDlg, WinCreateDlg, and WinDlgBox */
        /* in their pCreateParams parameter. */
        pParm = (MPARAM)mp2;

        /* Place the pointer in the window word area */
        WinSetWindowULong(hwndClient,
                          QWL_USER,
                          (ULONG) pParm);

    case WM_COMMAND:
        switch (SHORT1FROMMP(mp1))
        {
            case DID_OK:
                /* Retrieve the pointer from the window word area */
```

```
        pParm = (MPARAM)WinQueryWindowULong(hwndClient,
                                              QWL_USER);
    }
}
```

WinSetWindowULong - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSetWindowUShort

WinSetWindowUShort - Syntax

This function sets an unsigned, short integer value into the memory of the reserved window words.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwnd; /* Window handle. */
LONG      index; /* Zero-based index of the value to be set. */
USHORT    us; /* Unsigned, short integer value to store in the window words. */
BOOL      rc; /* Success indicator. */

rc = WinSetWindowUShort(hwnd, index, us);
```

WinSetWindowUShort Parameter - hwnd

hwnd ([HWND](#)) - input
Window handle.

WinSetWindowUShort Parameter - index

index ([LONG](#)) - input

Zero-based index of the value to be set.

The units of *index* are bytes. Valid values are zero through (*cbWindowData* -2), where *cbWindowData* is the parameter in [WinRegisterClass](#) that specifies the number of bytes available for application-defined storage. Any of the following QWS_* values are also valid.

Note: QWL_* values cannot be used.

QWS_CXRESTORE

The width to which the window is restored.

See also the QWS_CYRESTORE value.

QWS_CYRESTORE

The height to which the window is restored.

These values are only valid while the window is maximized or minimized (that is, while either the WS_MINIMIZED or WS_MAXIMIZED window style indicators are set). Changing these values with the WinSetWindowUShort call alters the restore size and position.

QWS_FLAGS

These indicators apply only to frame or dialog windows, and contain combinations of the following indicators:

FF_ACTIVE	Frame window is displayed in the active state.
FF_DIALOGBOX	Frame window is being used as a dialog box.
FF_DLGDISMISSED	Dialog has been dismissed by the WinDismissDlg function.
FF_FLASHHILITE	Window is currently flashed. This indicator toggles with each flash.
FF_FLASHWINDOW	Frame window is flashing.
FF_OWNERDISABLED	Window's owner is disabled. This indicator is only set if the window and its owner are siblings.
FF_OWNERHIDDEN	Frame window is hidden as a result of its owner being hidden or minimized. This indicator is set only if the window and its owner are siblings.
FF_SELECTED	Frame window is selected.

QWS_ID

Window identity. The value of the *id* parameter of the [WinCreateWindow](#) function.

QWS_RESULT

Dialog-result parameter, as established by the [WinDismissDlg](#) function.

QWS_XMINIMIZE

The x-coordinate of the position to which the window is minimized. If this value is -1, the window has not been minimized.

See also the QWS_YMINIMIZE value.

QWS_XRESTORE

The x-coordinate of the position to which the window is restored.

See also the QWS_CYRESTORE value.

QWS_YMINIMIZE

The y-coordinate of the position to which the window is minimized.

When the window is minimized for the first time an arbitrary position is chosen. Changing these values with the WinSetWindowUShort call alters the position of the minimized window, but only when the window is not in a minimized state.

QWS_YRESTORE

	The y-coordinate of the position to which the window is restored.
	See also the QWS_CYRESTORE value.
Other	Zero-based index.

WinSetWindowUShort Parameter - us

us ([USHORT](#)) - input
Unsigned, short integer value to store in the window words.

WinSetWindowUShort Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinSetWindowUShort - Parameters

hwnd ([HWND](#)) - input
Window handle.

index ([LONG](#)) - input
Zero-based index of the value to be set.

The units of *index* are bytes. Valid values are zero through (*cbWindowData* -2), where *cbWindowData* is the parameter in [WinRegisterClass](#) that specifies the number of bytes available for application-defined storage. Any of the following QWS_* values are also valid.

Note: QWL_* values cannot be used.

QWS_CXRESTORE
The width to which the window is restored.
See also the QWS_CYRESTORE value.

QWS_CYRESTORE
The height to which the window is restored.

These values are only valid while the window is maximized or minimized (that is, while either the WS_MINIMIZED or WS_MAXIMIZED window style indicators are set). Changing these values with the WinSetWindowUShort call alters the restore size and position.

QWS_FLAGS
These indicators apply only to frame or dialog windows, and contain combinations of the following indicators:

	FF_ACTIVE	Frame window is displayed in the active state.
	FF_DIALOGBOX	Frame window is being used as a dialog box.
	FF_DLGDISMISSED	Dialog has been dismissed by the WinDismissDlg function.
	FF_FLASHHILITE	Window is currently flashed. This indicator toggles with each flash.
	FF_FLASHWINDOW	Frame window is flashing.
	FF_OWNERDISABLED	Window's owner is disabled. This indicator is only set if the window and its owner are siblings.
	FF_OWNERHIDDEN	Frame window is hidden as a result of its owner being hidden or minimized. This indicator is set only if the window and its owner are siblings.
	FF_SELECTED	Frame window is selected.
QWS_ID	Window identity. The value of the <i>id</i> parameter of the WinCreateWindow function.	
QWS_RESULT	Dialog-result parameter, as established by the WinDismissDlg function.	
QWS_XMINIMIZE	The x-coordinate of the position to which the window is minimized. If this value is -1, the window has not been minimized. See also the QWS_YMINIMIZE value.	
QWS_XRESTORE	The x-coordinate of the position to which the window is restored. See also the QWS_CYRESTORE value.	
QWS_YMINIMIZE	The y-coordinate of the position to which the window is minimized. When the window is minimized for the first time an arbitrary position is chosen. Changing these values with the WinSetWindowUShort call alters the position of the minimized window, but only when the window is not in a minimized state.	
QWS_YRESTORE	The y-coordinate of the position to which the window is restored. See also the QWS_CYRESTORE value.	
Other	Zero-based index.	
us (USHORT) - input	Unsigned, short integer value to store in the window words.	
rc (BOOL) - returns	Success indicator.	
TRUE	Successful completion	
FALSE	Error occurred.	

WinSetWindowUShort - Related Functions

Related Functions

- [WinQueryWindowPtr](#)
- [WinQueryWindowULong](#)
- [WinQueryWindowUShort](#)
- [WinSetWindowBits](#)
- [WinSetWindowPtr](#)
- [WinSetWindowULong](#)
- [WinSetWindowUShort](#)

WinSetWindowUShort - Example Code

This example changes the height to which a window is restored to 100 by changing the value of a system defined window word.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HWND hwnd;

/* The height to which the window is restored */
WinSetWindowUShort(hwnd,
                    QWS_CYRESTORE,
                    (USHORT)100);
```

WinSetWindowUShort - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinShowCursor

WinShowCursor - Syntax

This function shows or hides the cursor that is associated with a specified window.

```
#define INCL_WINCursors /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND    hwnd; /* Handle of window to which the cursor belongs. */
BOOL    fShow; /* Show indicator. */
BOOL    rc; /* Success indicator. */
```

```
rc = WinShowCursor(hwnd, fShow);
```

WinShowCursor Parameter - hwnd

hwnd (**HWND**) - input
Handle of window to which the cursor belongs.

WinShowCursor Parameter - fShow

fShow (**BOOL**) - input
Show indicator.

TRUE	Make cursor visible
FALSE	Make cursor invisible.

WinShowCursor Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred, or an attempt was made to show the cursor when it was already visible.

WinShowCursor - Parameters

hwnd (**HWND**) - input
Handle of window to which the cursor belongs.

fShow (**BOOL**) - input
Show indicator.

TRUE	Make cursor visible
FALSE	Make cursor invisible.

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred, or an attempt was made to show the cursor when it was already visible.

WinShowCursor - Remarks

This function must be called by the same thread that created the cursor that is affected.

A cursor show-level count is maintained. It is incremented by a hide operation and decremented by a show operation. The cursor is actually visible if the cursor show-level count is zero, otherwise it is invisible. When decrementing, the cursor show-level count is fixed at zero so as not to show the cursor too many times, but it is possible to hide the cursor a number of times in succession.

WinShowCursor - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinShowCursor - Related Functions

Related Functions

- [WinCreateCursor](#)
 - [WinDestroyCursor](#)
 - [WinQueryCursorInfo](#)
 - WinShowCursor
-

WinShowCursor - Example Code

This example shows the cursor if it is successfully created.

```
#define INCL_WINCursors
#include <OS2.H>

HWND hwnd; /* handle of window that has pointer captured */
RECT rcl;

WinQueryWindowRect(hwnd, &rcl);

if (WinCreateCursor(hwnd, /* This must be the handle */
                    /* of a window for which */
                    /* the application can */
                    /* receive input. */
                    100, /* x,y position of cursor. */
                    100,
                    5, /* cursor width. */
                    5, /* cursor height. */) != 0)
```

```

        CURSOR_FLASH,
        &rc1)) /* region where the cursor */
                /* is visible. */
WinShowCursor(hwnd,
                TRUE); /* make cursor visible. */

```

WinShowCursor - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinShowPointer

WinShowPointer - Syntax

This function adjusts the pointer display level to show or hide a pointer.

```

#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndDesktop; /* Desktop-window handle. */
BOOL    fShow;        /* Level-update indicator. */
BOOL    rc;           /* Display-level-updated indicator. */

rc = WinShowPointer(hwndDesktop, fShow);

```

WinShowPointer Parameter - hwndDesktop

hwndDesktop (HWND) - input
Desktop-window handle.

HWND_DESKTOP
 The desktop-window handle

Other
 The specified desktop-window handle.

WinShowPointer Parameter - fShow

fShow (BOOL) - input	Level-update indicator.
TRUE	Decrement pointer display level by one. (The pointer level is not decremented to a negative value.)
FALSE	Increment pointer display level by one.

WinShowPointer Return Value - rc

rc (BOOL) - returns	Display-level-updated indicator.
TRUE	Pointer display level successfully updated.
FALSE	Pointer display level not successfully updated

WinShowPointer - Parameters

hwndDesktop (HWND) - input	Desktop-window handle.
HWND_DESKTOP	The desktop-window handle
Other	The specified desktop-window handle.
fShow (BOOL) - input	Level-update indicator.
TRUE	Decrement pointer display level by one. (The pointer level is not decremented to a negative value.)
FALSE	Increment pointer display level by one.
rc (BOOL) - returns	Display-level-updated indicator.
TRUE	Pointer display level successfully updated.
FALSE	Pointer display level not successfully updated

WinShowPointer - Remarks

The pointer display level determines whether the pointer is shown. If it is zero, the pointer is visible, but if it is greater than zero, the pointer is not visible. The initial setting of the pointer display level is dependent on the capabilities of the device. If a pointing device exists, the initial setting of the pointer display level is zero, otherwise it is one. The existing pointer display level can be obtained by using the [WinQuerySysValue](#) function with *iSysValue* set to SV_POINTERLEVEL.

WinShowPointer - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinShowPointer - Related Functions

Related Functions

- [WinCreatePointer](#)
 - [WinCreatePointerIndirect](#)
 - [WinDestroyPointer](#)
 - [WinDrawPointer](#)
 - [WinLoadPointer](#)
 - [WinQueryPointer](#)
 - [WinQueryPointerInfo](#)
 - [WinQueryPointerPos](#)
 - [WinQuerySysPointer](#)
 - [WinQuerySysPointerData](#)
 - [WinSetPointer](#)
 - [WinSetPointerPos](#)
 - [WinSetSysPointerData](#)
 - [WinShowPointer](#)
-

WinShowPointer - Example Code

This example obtains the pointer handle from the desktop window handle and hides the pointer.

```
#define INCL_WINPOINTERS
#define INCL_WINDESKTOP
#include <OS2.H>
HPOINTER hpointer;
HWND hwnd;

hpointer = WinQueryPointer(HWND_DESKTOP);

WinShowPointer(hwnd,FALSE);
```

WinShowPointer - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinShowTrackRect

WinShowTrackRect - Syntax

This function hides or shows the tracking rectangle.

```
#define INCL_WINTRACKRECT /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd;    /* Window handle. */
BOOL    fShow;   /* Show indicator. */
BOOL    rc;      /* Success indicator. */

rc = WinShowTrackRect(hwnd, fShow);
```

WinShowTrackRect Parameter - hwnd

hwnd ([HWND](#)) - input
Window handle.

Passed to the [WinTrackRect](#) function.

WinShowTrackRect Parameter - fShow

fShow ([BOOL](#)) - input
Show indicator.

TRUE	Show the tracking rectangle
FALSE	Hide the tracking rectangle.

WinShowTrackRect Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinShowTrackRect - Parameters

hwnd (**HWND**) - input
Window handle.

Passed to the [WinTrackRect](#) function.

fShow (**BOOL**) - input
Show indicator.

TRUE	Show the tracking rectangle
FALSE	Hide the tracking rectangle.

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinShowTrackRect - Remarks

This function maintains a show count. When a hide request is made, this count is decremented; when a show request is made, the count is incremented. When the count makes a transition from 0 to -1, the rectangle is hidden; when the count makes a transition from -1 to 0, the rectangle is shown.

When a rectangle is tracking, the application must call this function to hide the rectangle if there is a possibility of corrupting the tracking rectangle while drawing. The rectangle is shown afterwards. Because the structure is updated continuously, the application can examine the coordinates of the current tracking rectangle to determine whether temporary hiding is necessary.

The only case where an application needs to use this function is during asynchronous drawing. If an application is drawing on one thread, and issuing [WinTrackRect](#) on another, unwanted areas of tracking rectangle may be left behind. The drawing thread is therefore responsible for calling this function whenever tracking is in progress. The application must provide for communication between the two threads to ensure that if one thread is tracking, the drawing thread issues this function. This can be done with a *semaphore*.

WinShowTrackRect - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinShowTrackRect - Related Functions

Related Functions

- [WinShowTrackRect](#)
 - [WinTrackRect](#)
-

WinShowTrackRect - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Related Functions](#)
[Glossary](#)

WinShowWindow

WinShowWindow - Syntax

This function sets the visibility state of a window.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HWND     hwnd;           /* Window handle. */
BOOL     fNewVisibility; /* New visibility state. */
BOOL     rc;             /* Visibility changed indicator. */

rc = WinShowWindow(hwnd, fNewVisibility);
```

WinShowWindow Parameter - hwnd

hwnd (**HWND**) - input
Window handle.

WinShowWindow Parameter - fNewVisibility

fNewVisibility (**BOOL**) - input
New visibility state.

- TRUE Set window state visible
- FALSE Set window state invisible.

WinShowWindow Return Value - rc

rc (**BOOL**) - returns
Visibility changed indicator.

- TRUE Window visibility successfully changed
- FALSE Window visibility not successfully changed.

WinShowWindow - Parameters

hwnd (**HWND**) - input
Window handle.

fNewVisibility (**BOOL**) - input
New visibility state.

- TRUE Set window state visible
- FALSE Set window state invisible.

rc (**BOOL**) - returns
Visibility changed indicator.

- TRUE Window visibility successfully changed
- FALSE Window visibility not successfully changed.

WinShowWindow - Remarks

A window possesses a visibility state indicated by the WS_VISIBLE style bit. When the WS_VISIBLE style bit is set, the window is shown and subsequent drawing into the window is presented, unless that window is obscured by some other window, or at least one of the windows upward in the parent chain from *hwnd* does not have the WS_VISIBLE style.

When the WS_VISIBLE style bit is not set, the window is not shown ("hidden") and subsequent drawing into the window is not presented, even if that window is not obscured by another window.

If the value of the WS_VISIBLE style bit has been changed, the [WM_SHOW](#) message is sent to the window of *hwnd* before the function returns.

WinShowWindow - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinShowWindow - Related Functions

Related Functions

- [WinBeginPaint](#)
- [WinEnableWindowUpdate](#)
- [WinEndPaint](#)
- [WinExcludeUpdateRegion](#)
- [WinGetClipPS](#)
- [WinGetPS](#)
- [WinGetScreenPS](#)
- [WinInvalidateRect](#)
- [WinInvalidateRegion](#)
- [WinIsWindowShowing](#)
- [WinIsWindowVisible](#)
- [WinLockVisRegions](#)
- [WinOpenWindowDC](#)
- [WinQueryUpdateRect](#)
- [WinQueryUpdateRegion](#)
- [WinRealizePalette](#)
- [WinReleasePS](#)
- [WinShowWindow](#)
- [WinUpdateWindow](#)
- [WinValidateRect](#)
- [WinValidateRegion](#)

WinShowWindow - Related Messages

Related Messages

- [WM_SHOW](#)

WinShowWindow - Example Code

This example uses the WinShowWindow call to make a modeless dialog window visible.

```
#define INCL_WINWINDOWMGR
#define INCL_WINDIALOGS
#include <OS2.H>
#define DLG_MODELESS 900
    /* dialog procedure declaration. */
MRESULT EXPENTRY DlgProc( HWND hwndDlg, ULONG msg, MPARAM mp1, MPARAM mp2 );
HWND hwnd;

    hwnd = WinLoadDlg( HWND_DESKTOP,
                      HWND_OBJECT,
                      (PFNWP)DlgProc,
                      (HMODULE)NULL,
                      DLG_MODELESS,
                      NULL);

/*
DlgProc( HWND hwndDlg, ULONG msg, MPARAM mp1,
        MPARAM mp2 )
{
CASE USER_DEFINED:

*/
    WinShowWindow( hwnd,
                  TRUE ); /* show window. */
    WinSetFocus( HWND_DESKTOP, hwnd );
}
```

WinShowWindow - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinShutdownSystem

WinShutdownSystem - Syntax

The WinShutdownSystem function will close down the system.

```
#define INCL_WINWORKPLACE
#include <os2.h>
```

```
HAB    hab; /* Anchor-block handle. */
HMQ    hmq; /* Message-queue handle. */
BOOL    rc; /* Success indicator. */

rc = WinShutdownSystem(hab, hmq);
```

WinShutdownSystem Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinShutdownSystem Parameter - hmq

hmq (**HMQ**) - input
Message-queue handle.

WinShutdownSystem Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinShutdownSystem - Parameters

hab (**HAB**) - input
Anchor-block handle.

hmq (**HMQ**) - input
Message-queue handle.

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinShutdownSystem - Remarks

The WinShutdownSystem function will close all running applications and will then call DosShutdown.

Presentation Manager applications will receive a WM_SAVEAPPLICATION message prior to a WM_QUIT message.

When the system is restarted, all applications that were running when WinShutdownSystem was last called will be restarted.

WinShutdownSystem - Related Functions

Related Functions

- [WinCancelShutdown](#)
-

WinShutdownSystem - Example Code

This example performs an OS/2 System Shutdown from a program.

```
#define INCL_WINWORKPLACE
#define INCL_DOSFILEMGR
#define INCL_DOSERRORS
#include <os2.h>
#include <stdio.h>

int main(VOID)
{
    HAB    hab        = NULLHANDLE;    /* Window handle */
    HMq     hmq        = NULLHANDLE;    /* Message queue handle */
    BOOL    fSuccess   = 0;              /* Win API success indicator */

    hab = WinInitialize( 0 );
    hmq = WinCreateMsgQueue( hab, 0 );

    /* Prevent our program from hanging the shutdown.  If this call is
       omitted, the system will wait for us to do a WinDestroyMsgQueue. */

    fSuccess = WinCancelShutdown( hmq, TRUE );

    /* Shutdown the system! */

    printf("System Shutdown will now be attempted...\n");
    fSuccess = WinShutdownSystem( hab, hmq );

    if (!fSuccess) {
        return 1;
    } else {
        return NO_ERROR;
    } /* endif */
}
```

WinShutdownSystem - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinStartApp

WinStartApp - Syntax

This function starts an application.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndNotify; /* Notification-window handle. */
PPROGDETAILS pDetails; /* Program list structure. */
PSZ        pszParams; /* Input parameters for the application to be started. */
PVOID      pReserved; /* Start data. */
ULONG      ulOptions; /* Option indicators. */
HAPP       happ; /* Application handle. */

happ = WinStartApp(hwndNotify, pDetails, pszParams,
                  pReserved, ulOptions);
```

WinStartApp Parameter - hwndNotify

hwndNotify ([HWND](#)) - input
Notification-window handle.

A [WM_APPTERMINATENOTIFY](#) message is posted to this window, when the started application terminates.

WinStartApp Parameter - pDetails

pDetails ([PPROGDETAILS](#)) - input
Program list structure.

WinStartApp Parameter - pszParams

pszParams ([PSZ](#)) - input
Input parameters for the application to be started.

This specifies the command line parameters to be passed to this application when it starts.

WinStartApp Parameter - pReserved

pReserved ([PVOID](#)) - input
Start data.

Reserved, must be NULL.

WinStartApp Parameter - ulOptions

ulOptions ([ULONG](#)) - input
Option indicators.

If more than one option is selected, the values can be ORed together.

WinStartApp Return Value - happ

happ ([HAPP](#)) - returns
Application handle.

NULL	Application not started.
Other	Application handle.

WinStartApp - Parameters

hwndNotify ([HWND](#)) - input
Notification-window handle.

A [WM_APPTERMINATENOTIFY](#) message is posted to this window, when the started application terminates.

pDetails ([PPROGDETAILS](#)) - input

Program list structure.

pszParams ([PSZ](#)) - input

Input parameters for the application to be started.

This specifies the command line parameters to be passed to this application when it starts.

pReserved ([PVOID](#)) - input

Start data.

Reserved, must be NULL.

ulOptions ([ULONG](#)) - input

Option indicators.

If more than one option is selected, the values can be ORed together.

happ ([HAPP](#)) - returns

Application handle.

NULL

Application not started.

Other

Application handle.

WinStartApp - Remarks

Starts the application identified in [PROGDETAILS](#). If the `ulOptions` parameter has the `SAF_INSTALLED_CMDLINE` set, the `pszParameters` in the `PROGDETAILS` structure is used; otherwise the `pszParams` passed in this function is used.

If the application is successfully started, the return value is a handle to the application. If `SAF_STARTCHILDAPP` is specified, this can be used to stop the application (see [WinTerminateApp](#)).

When the program specified by the application handle terminates, the window specified by the *hwndNotify* parameter (if the window still exists and is valid) has a [WM_APP_TERMINATENOTIFY](#) message posted to it to notify it of the application termination.

This function requires the existence of a message queue.

If Dos/Windows applications have to be started with `WinStartApp`, Global settings from the `USERPROFILE` will have to be read and passed in the [PROGDETAILS](#) structure in the *pszEnvironment* variable. If not done, `WinStartApp` will take the system default values and start the application.

WinStartApp - Errors

Possible returns from [WinGetLastError](#)

`PMERR_DOS_ERROR` (0x1200)

A DOS call returned an error.

`PMERR_INVALID_APPL` (0x1530)

Attempted to start an application whose type is not recognized by OS/2.

`PMERR_INVALID_PARAMETERS` (0x1208)

An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32 768 to +32 767 cannot be converted to a [SHORT](#), and a negative number cannot be converted to a [ULONG](#) or [USHORT](#).

`PMERR_INVALID_WINDOW` (0x1206)

The window specified with a Window List call is not a valid frame window.

`PMERR_STARTED_IN_BACKGROUND` (0x1532)

The application started a new session in the background.

WinStartApp - Related Functions

Related Functions

- [WinTerminateApp](#)
-

WinStartApp - Related Messages

Related Messages

- [WM_APPTERMINATENOTIFY](#)
-

WinStartApp - Example Code

This example calls WinStartApp in a typical termination sequence.

```
#define INCL_DOSSESMGR
#include <os2.h>

HWND      hwndNotify;
PPROGDETAILS pDetails;
HAPP      happ;

pDetails = (PPROGDETAILS) malloc( sizeof(PROGDETAILS) ); /* Allocate structure */

pDetails->Length                = sizeof(PROGDETAILS);
pDetails->progt.progc            = PROG_WINDOWABLEVIO;
pDetails->progt.fbVisible        = SHE_VISIBLE;
pDetails->pszTitle               = "TEXT";
pDetails->pszExecutable          = "TEXT.EXE";
pDetails->pszParameters          = NULL;
pDetails->pszStartupDir          = "";
pDetails->pszICON                = "T.ICO";
pDetails->pszEnvironment         = "WORKPLACE\0\0";
pDetails->swpInitial.fl          = SWP_ACTIVATE;          /* Window positioning */
pDetails->swpInitial.cy          = 0;                      /* Width of window */
pDetails->swpInitial.cx          = 0;                      /* Height of window */
pDetails->swpInitial.y           = 0;                      /* Lower edge of window */
pDetails->swpInitial.x           = 0;                      /* Left edge of window */
pDetails->swpInitial.hwndInsertBehind = HWND_TOP;
pDetails->swpInitial.hwnd        = hwndNotify;
pDetails->swpInitial.ulReserved1 = 0;
pDetails->swpInitial.ulReserved2 = 0;

happ = WinStartApp(hwndNotify,
                   pDetails,
                   NULL, NULL,
                   SAF_STARTCHILDAPP);
.
.
.
WinTerminateApp(happ);
```

The following example calls WinStartApp inorder to start Dos/Windows applications.

```
#define INCL_WIN
#define INCL_DOSSESMGR
```

```

#include <os2.h>

PROGDETAILS  pDetails;
HAPP         happ;
ULONG        dataLen=0;
ULONG        length=0;
BOOL         rc;
PSZ          pBuffer;

/*Query for the size of the settings for a particular key */
rc = PrfQueryProfileSize(HINI_USERPROFILE,"WINOS2","PM_GlobalWindows31Settings",
    if(rc  dataLen > 0)
    {
        pBuffer = (PSZ)malloc((DataLen+1)*sizeof(char));
        if(pBuffer)
        {
            memset(pBuffer,0,(DataLen+1)*sizeof(char));
            /*Now get the settings */
            length = PrfQueryProfileString(HINI_USERPROFILE,"WINOS2","PM_GlobalWindows31Settings",NULL,pBuffer,dat

            /* The settings retrieved from USERPROFILE are seperated with a ';'. These will have to be replaced
               and an additional '\0' will have to be inserted into tthe string at the end*/

            if(length > 0)
            {
                pDetails.Length          = sizeof(PROGDETAILS);
                pDetails.progt.progc     = PROG_31_STDSEAMLESSCOMMON;
                pDetails.progt.fbVisible = SHE_VISIBLE;
                pDetails.pszTitle        = "Calculator";
                pDetails.pszExecutable   = "calc.exe";
                pDetails.pszParameters   = NULL;
                pDetails.pszStartupDir    = NULL;
                pDetails.pszIcon          = NULL;
                pDetails.pszEnvironment = pBuffer;
                pDetails.swpInitial.fl    = SWP_SHOW;
                pDetails.swpInitial.cy    = 0;
                pDetails.swpInitial.cx    = 0;
                pDetails.swpInitial.y     = 0;
                pDetails.swpInitial.x     = 0;
                pDetails.swpInitial.hwndInsertBehind = HWND_TOP;
                pDetails.swpInitial.hwnd  = (HWND)NULL;
                pDetails.swpInitial.ulReserved1 = 0;
                pDetails.swpInitial.ulReserved2 = 0;

                happ = WinStartApp((HWND)NULL, INSTALLED cmdline);
            }
        }
    }
}

```

WinStartApp - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Errors](#)
- [Remarks](#)
- [Example Code](#)
- [Related Functions](#)
- [Related Messages](#)
- [Glossary](#)

WinStartTimer

WinStartTimer - Syntax

This function starts a timer.

```
#define INCL_WINTIMER /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;           /* Anchor-block handle. */
HWND     hwnd;          /* Window handle that is part of the timer identification. */
ULONG     idTimer;       /* Timer identifier. */
ULONG     dtTimeout;     /* Delay time in milliseconds. */
ULONG     idTimerStarted; /* Timer identity. */

idTimerStarted = WinStartTimer(hab, hwnd,
                               idTimer, dtTimeout);
```

WinStartTimer Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinStartTimer Parameter - hwnd

hwnd (**HWND**) - input
Window handle that is part of the timer identification.

NULLHANDLE

The *idTimer* parameter is ignored, and this function returns a unique, nonzero, identity which represents that timer. The timer message is posted in the queue associated with the current thread, with the parameter of the **QMSG** structure set to NULLHANDLE.

Other

Window handle.

WinStartTimer Parameter - idTimer

idTimer (**ULONG**) - input
Timer identifier.

The value of an application-timer identifier must be below TID_USERMAX to avoid clashes with timers used by the system.

A timer identification, TID_SCROLL, is created by a scroll bar control. An application does not normally see the associated WM_TIMER, but passes it to the scroll-bar control.

A timer identification, TID_CURSOR, is created when the cursor is flashing. An application must ensure that the associated WM_TIMER is passed on to the default window procedure.

WinStartTimer Parameter - dtTimeout

dtTimeout (ULONG) - input
Delay time in milliseconds.

For OS/2 Warp Version 3, the value of this parameter must be in the range of 0-4 294 967 295.

For OS/2 2.1 and earlier, the value of this parameter must be in the range of 0-65 535.

WinStartTimer Return Value - idTimerStarted

idTimerStarted (ULONG) - returns
Timer identity.

A return value of 0 indicates that an error occurred.

WinStartTimer - Parameters

hab (HAB) - input
Anchor-block handle.

hwnd (HWND) - input
Window handle that is part of the timer identification.

NULLHANDLE

The *idTimer* parameter is ignored, and this function returns a unique, nonzero, identity which represents that timer. The timer message is posted in the queue associated with the current thread, with the parameter of the QMSG structure set to NULLHANDLE.

Other

Window handle.

idTimer (ULONG) - input
Timer identifier.

The value of an application-timer identifier must be below TID_USERMAX to avoid clashes with timers used by the system.

A timer identification, TID_SCROLL, is created by a scroll bar control. An application does not normally see the associated WM_TIMER, but passes it to the scroll-bar control.

A timer identification, TID_CURSOR, is created when the cursor is flashing. An application must ensure that the associated WM_TIMER is passed on to the default window procedure.

dtTimeout (ULONG) - input
Delay time in milliseconds.

For OS/2 Warp Version 3, the value of this parameter must be in the range of 0-4 294 967 295.

For OS/2 2.1 and earlier, the value of this parameter must be in the range of 0-65 535.

idTimerStarted ([ULONG](#)) - returns
Timer identity.

A return value of 0 indicates that an error occurred.

WinStartTimer - Remarks

This function creates a timer identified by *hwnd* and *idTimer*, set to time out every *dtTimeout* milliseconds. When a timer times out, a [WM_TIMER](#) message is posted.

A *dtTimeout* value of zero causes the timer to timeout as fast as possible; generally, this is about 1/18 second.

A second call to this function, for a timer that already exists, resets that timer.

WinStartTimer - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinStartTimer - Related Functions

Related Functions

- [WinGetCurrentTime](#)
 - [WinQueryMsgTime](#)
 - [WinStartTimer](#)
 - [WinStopTimer](#)
-

WinStartTimer - Related Messages

Related Messages

- [WM_TIMER](#)
-

WinStartTimer - Example Code

This example uses the WinStartTimer call to add up elapsed seconds.

```
#define INCL_WINTIMER
#include <OS2.H>

HAB hab; /* Anchor-block handle */
ULONG seconds;
ULONG msg;

WinStartTimer(hab,
              (HWND)0,
              0, /* Ignored as previous parm. is null */
              1000UL);

switch(msg)
{
    case WM_TIMER:
        seconds += 1;
        break;
}
```

WinStartTimer - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Errors](#)
- [Remarks](#)
- [Example Code](#)
- [Related Functions](#)
- [Related Messages](#)
- [Glossary](#)

WinStopTimer

WinStopTimer - Syntax

This function stops a timer.

```
#define INCL_WINTIMER /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB hab; /* Anchor-block handle. */
HWND hwnd; /* Window handle. */
ULONG ulTimer; /* Timer identifier. */
BOOL rc; /* Success indicator. */

rc = WinStopTimer(hab, hwnd, ulTimer);
```

WinStopTimer Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinStopTimer Parameter - hwnd

hwnd (**HWND**) - input
Window handle.

WinStopTimer Parameter - ulTimer

ulTimer (**ULONG**) - input
Timer identifier.

WinStopTimer Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred, or timer did not exist.

WinStopTimer - Parameters

hab (**HAB**) - input
Anchor-block handle.

hwnd (**HWND**) - input
Window handle.

ulTimer (**ULONG**) - input
Timer identifier.

rc (**BOOL**) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred, or timer did not exist.

WinStopTimer - Remarks

When this function is called, no further messages are received from the stopped timer, even if it has timed out since the last call to [WinGetMsg](#).

WinStopTimer - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinStopTimer - Related Functions

Related Functions

- [WinGetCurrentTime](#)
- [WinQueryMsgTime](#)
- [WinStartTimer](#)
- WinStopTimer

WinStopTimer - Example Code

This example uses the WinStopTimer call to stop a clock after one minute.

```
#define INCL_WINTIMER
#include <OS2.H>
HAB hab;                /* anchor-block handle. */
ULONG ulTimerId;
HWND hwnd;
ulTimerId = WinStartTimer(hab,
                          (HWND)0,
                          0, /* ignored because previous parameter */
                          /* is null. */                          /*
                          1000UL);

BOOL WndProc(. . . . ) {
static ULONG seconds;

switch(msg)
{
case WM_TIMER:
if (seconds) {
```

```

        seconds ++ ;
        if (seconds == 60) WinStopTimer(hab, hwnd, ulTimerId);
    }
    break;
case WM_CREATE:
    seconds = 0;
    .
    .
    .
}

```

WinStopTimer - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinStoreWindowPos

WinStoreWindowPos - Syntax

The WinStoreWindowPos function will save the current size and position of the window specified by *hwnd*.

```

#define INCL_WINWORKPLACE
#include <os2.h>

PSZ      pAppName; /* Pointer to application name. */
PSZ      pKeyName; /* Pointer to key name. */
HWND     hwnd;     /* Window handle for the window to be stored. */
BOOL     rc;       /* Success indicator. */

rc = WinStoreWindowPos(pAppName, pKeyName,
    hwnd);

```

WinStoreWindowPos Parameter - pAppName

pAppName ([PSZ](#)) - input

Pointer to application name.

A pointer to a zero-terminated string which contains the application name.

WinStoreWindowPos Parameter - pKeyName

pKeyName (PSZ) - input

Pointer to key name.

A pointer to a zero-terminated string which contains the key name.

WinStoreWindowPos Parameter - hwnd

hwnd (HWND) - input

Window handle for the window to be stored.

WinStoreWindowPos Return Value - rc

rc (BOOL) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinStoreWindowPos - Parameters

pAppName (PSZ) - input

Pointer to application name.

A pointer to a zero-terminated string which contains the application name.

pKeyName (PSZ) - input

Pointer to key name.

A pointer to a zero-terminated string which contains the key name.

hwnd (HWND) - input

Window handle for the window to be stored.

rc (BOOL) - returns

Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinStoreWindowPos - Remarks

This function will also save the presentation parameters.

WinStoreWindowPos - Related Functions

Related Functions

- [WinRestoreWindowPos](#)

WinStoreWindowPos - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Related Functions](#)
- [Glossary](#)

WinStretchPointer

WinStretchPointer - Syntax

This function draws a pointer in the passed *hps* at the passed coordinates [*lx*, *ly*] and at the passed size [*lcx*, *lcy*].

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HPS      hps;          /* Presentation space handle into which the pointer is drawn. */
LONG     lx;           /* X coordinate at which to draw the pointer, in device coordinates. */
LONG     ly;           /* Y coordinate at which to draw the pointer, in device coordinates. */
LONG     lcx;          /* X size at which to draw the pointer, in device coordinates. */
LONG     lcy;          /* Y size at which to draw the pointer, in device coordinates. */
HPOINTER hptrPointer;  /* Pointer handle. */
ULONG    ulHalftone;   /* Shading control with which to draw the pointer: */
```

```
BOOL          fSuccess;          /* Success indicator: */  
  
fSuccess = WinStretchPointer(hps, lx, ly,  
                             lcx, lcy, hptrPointer, ulHalftone);
```

WinStretchPointer Parameter - hps

hps (**HPS**) - input

Presentation space handle into which the pointer is drawn.

This can be either a micro presentation space or a normal presentation space (see GpiCreatePS).

WinStretchPointer Parameter - lx

lx (**LONG**) - input

X coordinate at which to draw the pointer, in device coordinates.

WinStretchPointer Parameter - ly

ly (**LONG**) - input

Y coordinate at which to draw the pointer, in device coordinates.

WinStretchPointer Parameter - lcx

lcx (**LONG**) - input

X size at which to draw the pointer, in device coordinates.

WinStretchPointer Parameter - lcy

lcy (**LONG**) - input

Y size at which to draw the pointer, in device coordinates.

WinStretchPointer Parameter - hptrPointer

hptrPointer ([HPOINTER](#)) - input
Pointer handle.

The pointer should be loaded using [WinLoadPointer](#), [WinCreatePointer](#), or [WinCreatePointerIndirect](#).

WinStretchPointer Parameter - ulHalftone

ulHalftone ([ULONG](#)) - input
Shading control with which to draw the pointer:

DP_NORMAL	As it normally appears.
DP_HALFTONED	With a halftone pattern where black normally appears.
DP_INVERTED	Inverted-black for white, and white for black.

WinStretchPointer Return Value - fSuccess

fSuccess ([BOOL](#)) - returns
Success indicator:

TRUE	Successful completion.
FALSE	The function failed.

WinStretchPointer - Parameters

hps ([HPS](#)) - input
Presentation space handle into which the pointer is drawn.

This can be either a micro presentation space or a normal presentation space (see [GpiCreatePS](#)).

lx ([LONG](#)) - input
X coordinate at which to draw the pointer, in device coordinates.

ly ([LONG](#)) - input
Y coordinate at which to draw the pointer, in device coordinates.

lcx ([LONG](#)) - input
X size at which to draw the pointer, in device coordinates.

lcy ([LONG](#)) - input
Y size at which to draw the pointer, in device coordinates.

hptrPointer ([HPOINTER](#)) - input
Pointer handle.

The pointer should be loaded using [WinLoadPointer](#), [WinCreatePointer](#), or [WinCreatePointerIndirect](#).

ulHalftone ([ULONG](#)) - input

Shading control with which to draw the pointer:

DP_NORMAL	As it normally appears.
DP_HALFTONED	With a halftone pattern where black normally appears.
DP_INVERTED	Inverted-black for white, and white for black.

fSuccess ([BOOL](#)) - returns

Success indicator:

TRUE	Successful completion.
FALSE	The function failed.

WinStretchPointer - Remarks

This function should only be used in draw mode (DM_DRAW) to a screen device context.

The function checks the supplied pointer to determine whether it contains a pointer format matching the size passed. If there is no such match the default format is stretched to size.

WinStretchPointer - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HPTR (0x101B)
An invalid pointer handle was specified.

PMERR_INVALID_FLAG (0x1019)
An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

WinStretchPointer - Related Functions

Related Functions

- [WinCreatePointer](#)
- [WinCreatePointerIndirect](#)
- [WinDestroyPointer](#)
- [WinDrawPointer](#)
- [WinLoadPointer](#)
- [WinQueryPointer](#)
- [WinQueryPointerInfo](#)
- [WinQueryPointerPos](#)
- [WinQuerySysPointer](#)
- [WinQuerySysPointerData](#)
- [WinSetPointer](#)
- [WinSetPointerPos](#)
- [WinSetSysPointerData](#)

- [WinShowPointer](#)

WinStretchPointer - Example Code

This example draws a pointer loaded using WinLoadPointer in response to a paint message (WM_PAINT).

```
#define INCL_WINPOINTERS                /* Window Pointer functions */
#include <os2.h>

HPS      hps;                          /* Presentation-space handle */
HWND     hwnd;                         /* Window handle             */
HPOINTER hptr;                         /* Pointer handle            */
BOOL     fSuccess;                    /* Success indicator         */
ULONG    ulHalftone=DP_NORMAL;        /* Draw with normal shading  */
ULONG    idPointer=ID_MYPOINTER;      /* Identifier of pointer in  */
                                           /* executable resources      */

case WM_CREATE:
    hptr = WinLoadPointer(HWND_DESKTOP, NULLHANDLE, idPointer);

case WM_PAINT:
    hps = WinBeginPaint(hwnd, NULLHANDLE, NULL);
    WinStretchPointer(hps, 100, 100, 16, 16, hptr, ulHalftone);
    WinEndPaint(hps);
```

WinStretchPointer - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSubclassWindow

WinSubclassWindow - Syntax

This function subclasses the indicated window by replacing its window procedure with another window procedure, specified by *pNewWindowProc*.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwnd;          /* Handle of window that is being subclassed. */
PFNWP     pNewWindowProc; /* New window procedure. */
PFNWP     pOldWindowProc; /* Old window procedure. */

pOldWindowProc = WinSubclassWindow(hwnd, pNewWindowProc);
```

WinSubclassWindow Parameter - hwnd

hwnd (**HWND**) - input
Handle of window that is being subclassed.

WinSubclassWindow Parameter - pNewWindowProc

pNewWindowProc (**PFNWP**) - input
New window procedure.

Window procedure used to subclass *hwnd*.

WinSubclassWindow Return Value - pOldWindowProc

pOldWindowProc (**PFNWP**) - returns
Old window procedure.

Previous window procedure belonging to *hwnd*.

If this function fails, 0L is returned.

WinSubclassWindow - Parameters

hwnd (**HWND**) - input
Handle of window that is being subclassed.

pNewWindowProc (**PFNWP**) - input
New window procedure.

Window procedure used to subclass *hwnd*.

pOldWindowProc (**PFNWP**) - returns
Old window procedure.

Previous window procedure belonging to *hwnd*.

If this function fails, 0L is returned.

WinSubclassWindow - Remarks

To subclass a window effectively, the new window procedure calls the old window procedure rather than [WinDefWindowProc](#), for those messages it does not process itself.

To reverse the effect of subclassing, call this function again using the old window procedure address.

Note: It is not possible to subclass a window created by another process.

WinSubclassWindow - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinSubclassWindow - Related Functions

Related Functions

- [WinCalcFrameRect](#)
- [WinCreateFrameControls](#)
- [WinCreateStdWindow](#)
- [WinCreateWindow](#)
- [WinDefWindowProc](#)
- [WinDestroyWindow](#)
- [WinQueryClassInfo](#)
- [WinQueryClassName](#)
- [WinRegisterClass](#)
- [WinSubclassWindow](#)

WinSubclassWindow - Example Code

This example uses the WinSubclassWindow call to subclass the frame window procedure, so that frame-sizing restrictions can be implemented.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
HAB hab;
PFNWP FrameWndProc, OldpFrame;
HWND hwndFrame;
```

```
OldpFrame = WinSubclassWindow(hwndFrame,  
                                (PFNWP)FrameWndProc);
```

```
MRESULT EXPENTRY FrameWndProc(hwnd, msg, mp1, mp2)  
{  
    .  
    .  
    .  
    switch(msg) {  
        case . . . :  
            .  
            .  
            .  
        default:  
            OldpFrame(hwnd, msg, mp1, mp2);  
    }  
}
```

WinSubclassWindow - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSubstituteStrings

WinSubstituteStrings - Syntax

This function performs a substitution process on a text string, replacing specific marker characters with text supplied by the application.

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, */  
#include <os2.h>  
  
HWND    hwnd;        /* Handle of window that processes the call. */  
PSZ     pszSrc;       /* Source string. */  
LONG    lDestMax;     /* Maximum number of characters returnable. */  
PSZ     pszDest;      /* Resultant string. */  
LONG    lDestRet;     /* Actual number of characters returned. */  
  
lDestRet = WinSubstituteStrings(hwnd, pszSrc,  
                                lDestMax, pszDest);
```

WinSubstituteStrings Parameter - hwnd

hwnd ([HWND](#)) - input
Handle of window that processes the call.

WinSubstituteStrings Parameter - pszSrc

pszSrc ([PSZ](#)) - input
Source string.

This is the text string that is to have substitution performed.

WinSubstituteStrings Parameter - IDestMax

IDestMax ([LONG](#)) - input
Maximum number of characters returnable.

This is the maximum number of characters that can be returned in *pszDest*. It must be greater than 0.

WinSubstituteStrings Parameter - pszDest

pszDest ([PSZ](#)) - output
Resultant string.

This is the text string produced by the substitution process.

The string is truncated if it would otherwise contain more than *IDestMax* characters. When truncation occurs, the last character of the truncated string is always the null-termination character.

WinSubstituteStrings Return Value - IDestRet

IDestRet ([LONG](#)) - returns
Actual number of characters returned.

This is the actual number returned in *pszDest*, excluding the null-termination character. The maximum value is (*IDestMax*-1). It is zero if an error occurred.

WinSubstituteStrings - Parameters

hwnd ([HWND](#)) - input
Handle of window that processes the call.

pszSrc ([PSZ](#)) - input
Source string.

This is the text string that is to have substitution performed.

IDestMax ([LONG](#)) - input
Maximum number of characters returnable.

This is the maximum number of characters that can be returned in *pszDest*. It must be greater than 0.

pszDest ([PSZ](#)) - output
Resultant string.

This is the text string produced by the substitution process.

The string is truncated if it would otherwise contain more than *IDestMax* characters. When truncation occurs, the last character of the truncated string is always the null-termination character.

IDestRet ([LONG](#)) - returns
Actual number of characters returned.

This is the actual number returned in *pszDest*, excluding the null-termination character. The maximum value is (*IDestMax*-1). It is zero if an error occurred.

WinSubstituteStrings - Remarks

When a string of the form "%n" (where n is in the range 0 through 9) occurs in the source string, a [WM_SUBSTITUTESTRING](#) message is sent to the specified window. This message returns a text string to use as a substitution for "%n" in the destination string, which is otherwise an exact copy of the source string.

If "%%" occurs in the source, "%" is copied to the destination, but no other substitution occurs. If "%x" occurs in the source, where x is not a digit or "%", the source is copied unchanged to the destination. The source and destination strings must not overlap in memory.

This function is particularly useful for displaying variable information in dialogs, menus, and other user-interface calls. Variable information can include such things as file names, which cannot be statically declared within resource files.

This function is called by the system while creating child windows in a dialog box. It allows the child windows to perform textual substitutions in their window text.

WinSubstituteStrings - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinSubstituteStrings - Related Functions

Related Functions

- [WinCompareStrings](#)
- [WinLoadString](#)
- [WinNextChar](#)
- [WinPrevChar](#)
- [WinSubstituteStrings](#)
- [WinUpper](#)
- [WinUpperChar](#)

WinSubstituteStrings - Related Messages

Related Messages

- [WM_SUBSTITUTESTRING](#)

WinSubstituteStrings - Example Code

This example shows how the substitution process works when the WinSubstituteStrings call is made.

```
#define INCL_WINDIALOGS
#include <OS2.H>

static MRESULT ClientWindowProc(HWND hwnd,ULONG msg,MPARAM mp1,MPARAM mp2);
test()
{
    HWND hwnd;
    char source[] = "This is the source string: %1";
    char result[22];
    MPARAM mp1;
    ULONG msg;

    /* This function performs a substitution process on a text string, */
    /* replacing specific marker characters with text supplied          */
    /* by the application.                                           */
    WinSubstituteStrings(hwnd,
                          source,
                          sizeof(result),
                          result);

    /* WM_SUBSTITUTESTRING message is sent to the window */
    /* defined by hwnd.                                   */
}
static MRESULT ClientWindowProc( HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2 )
{
    switch(msg)
    {
        case WM_SUBSTITUTESTRING:
            switch( (ULONG)mp1)
            {
                case 1:
                    return(MRFROMP("A"));
                    break;
            }
            break;
    }
}
```

WinSubstituteStrings - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinSubtractRect

WinSubtractRect - Syntax

This function subtracts one rectangle from another.

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
PRECTL   prclDest;     /* Result. */
PRECTL   prclSrc1;     /* First source rectangle. */
PRECTL   prclSrc2;     /* Second source rectangle. */
BOOL      rc;          /* Not-empty indicator. */

rc = WinSubtractRect(hab, prclDest, prclSrc1,
                    prclSrc2);
```

WinSubtractRect Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinSubtractRect Parameter - prclDest

prclDest (**PRECTL**) - output
Result.

The result of the subtraction of *prclSrc2* from *prclSrc1*.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type **WRECT** may also be used, if supported by the language.

WinSubtractRect Parameter - prclSrc1

prclSrc1 (**PRECTL**) - input
First source rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type **WRECT** may also be used, if supported by the language.

WinSubtractRect Parameter - prclSrc2

prclSrc2 (**PRECTL**) - input
Second source rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type **WRECT** may also be used, if supported by the language.

WinSubtractRect Return Value - rc

rc (**BOOL**) - returns
Not-empty indicator.

TRUE	Rectangle is not empty
FALSE	Rectangle is empty or an error occurred.

WinSubtractRect - Parameters

hab (**HAB**) - input
Anchor-block handle.

prclDest (**PRECTL**) - output
Result.

The result of the subtraction of *prcSrc2* from *prcSrc1*.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) may also be used, if supported by the language.

prcSrc1 ([PRECTL](#)) - input
First source rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) may also be used, if supported by the language.

prcSrc2 ([PRECTL](#)) - input
Second source rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) may also be used, if supported by the language.

rc ([BOOL](#)) - returns
Not-empty indicator.

TRUE	Rectangle is not empty
FALSE	Rectangle is empty or an error occurred.

WinSubtractRect - Remarks

Subtracts *prcSrc2* from *prcSrc1*.

prcSrc1, *prcSrc2*, and *prcDest* must be distinct [RECTL](#) structures.

Subtracting one rectangle from another does not always result in a rectangular area. When this occurs, this function returns *prcSrc1* in *prcDest*. For this reason, this function provides only an approximation of subtraction. However, the area described by *prcDest* is always greater than, or equal to, the true result of the subtraction.

GpiCombineRegion can be used to calculate the true result of the subtraction of two rectangular areas. The WinSubtractRect function is much faster.

WinSubtractRect - Related Functions

Related Functions

- [WinCopyRect](#)
- [WinEqualRect](#)
- [WinFillRect](#)
- [WinInflateRect](#)
- [WinIntersectRect](#)
- [WinIsRectEmpty](#)
- [WinOffsetRect](#)
- [WinPtInRect](#)
- [WinSetRect](#)
- [WinSetRectEmpty](#)
- [WinSubtractRect](#)
- [WinUnionRect](#)

WinSubtractRect - Example Code

This example uses the WinSubtractRect call to subtract two rectangles.

```
#define INCL_WINRECTANGLES
#include <OS2.H>
HAB hab;
RECTL resultrcl; /* result. */
RECTL rclminuend={25, /* x coordinate of left-hand edge of */
                  /* rectangle. */
                  25, /* y coordinate of bottom edge of */
                  /* rectangle. */
                  425,/* x coordinate of right-hand edge of */
                  /* rectangle. */
                  425};/* y coordinate of top edge of rectangle. */

RECTL rclsubtrahend={15, /* x coordinate of left-hand edge of */
                    /* rectangle. */
                    15, /* y coordinate of bottom edge of */
                    /* rectangle. */
                    125,/* x coordinate of right-hand edge of */
                    /* rectangle. */
                    125};/* y coordinate of top edge of rectangle.*/

WinSubtractRect(hab,
                &resultrcl,
                &rclminuend,
                &rclsubtrahend);
```

WinSubtractRect - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinSwitchToProgram

WinSwitchToProgram - Syntax

This function makes a specific program the active program.

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HSWITCH    hswitchSwHandle; /* Window List entry handle of program to be activated. */
ULONG      rc;              /* Return code. */
```

```
rc = WinSwitchToProgram(hswitchSwHandle);
```

WinSwitchToProgram Parameter - hswitchSwHandle

hswitchSwHandle ([H SWITCH](#)) - input
Window List entry handle of program to be activated.

WinSwitchToProgram Return Value - rc

rc ([ULONG](#)) - returns
Return code.

0
Successful completion.

INV_SWITCH_LIST_ENTRY_HANDLE
Invalid Window List entry handle of the program to be activated.

NOT_PERMITTED_TO_CAUSE_SWITCH
Requesting program is not the current foreground process.

WinSwitchToProgram - Parameters

hswitchSwHandle ([H SWITCH](#)) - input
Window List entry handle of program to be activated.

rc ([ULONG](#)) - returns
Return code.

0
Successful completion.

INV_SWITCH_LIST_ENTRY_HANDLE
Invalid Window List entry handle of the program to be activated.

NOT_PERMITTED_TO_CAUSE_SWITCH
Requesting program is not the current foreground process.

WinSwitchToProgram - Remarks

Use of this function causes another window (and its related windows) of a PM session to appear on the front of the screen, or a switch to another session in the case of a non-PM program. In either case, the keyboard (and mouse for the non-PM case) input is directed to the new program.

WinSwitchToProgram - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_SWITCH_HANDLE (0x1202)
An invalid Window List entry handle was specified.

WinSwitchToProgram - Related Functions

Related Functions

- [WinAddSwitchEntry](#)
 - [WinChangeSwitchEntry](#)
 - [WinCreateSwitchEntry](#)
 - [WinQuerySessionTitle](#)
 - [WinQuerySwitchEntry](#)
 - [WinQuerySwitchHandle](#)
 - [WinQuerySwitchList](#)
 - [WinQueryTaskSizePos](#)
 - [WinQueryTaskTitle](#)
 - [WinRemoveSwitchEntry](#)
 - [WinSwitchToProgram](#)
-

WinSwitchToProgram - Example Code

This example calls WinSwitchToProgram to make a window the foreground process.

```
#define INCL_WINSWITCHLIST
#include <OS2.H>

HAB      hab;
HWND     hwndFrame;
HSWITCH  hswitch;

hswitch = WinQuerySwitchHandle(hwndFrame, 0);

/* Switch to the window defined by hwndFrame */
WinSwitchToProgram(hswitch);
```

WinSwitchToProgram - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)

WinTerminate

WinTerminate - Syntax

This function terminates an application thread's use of the Presentation Manager and releases all of its associated resources.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMMON section */
#include <os2.h>

HAB      hab; /* Anchor-block handle. */
BOOL     rc;  /* Termination indicator. */

rc = WinTerminate(hab);
```

WinTerminate Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinTerminate Return Value - rc

rc ([BOOL](#)) - returns
Termination indicator.

TRUE	Application usage of Presentation Manager successfully terminated
FALSE	Application usage of Presentation Manager not successfully terminated, or WinInitialize has not been issued on this thread.

WinTerminate - Parameters

hab ([HAB](#)) - input

Anchor-block handle.

rc ([BOOL](#)) - returns
Termination indicator.

TRUE

Application usage of Presentation Manager successfully terminated

FALSE

Application usage of Presentation Manager not successfully terminated, or [WinInitialize](#) has not been issued on this thread.

WinTerminate - Remarks

It is good practice to issue this function before terminating an application thread. Before issuing this function, the application must destroy all windows and message queues that have been created by the thread, and return any cached presentation spaces to the cache. If it does not do so, the results, and the return value from this and subsequent calls are indeterminate.

WinTerminate - Related Functions

Related Functions

- [WinCancelShutdown](#)
- [WinCreateMsgQueue](#)
- [WinInitialize](#)
- WinTerminate

WinTerminate - Example Code

This example calls WinTerminate in a typical termination sequence.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
HAB hab;
HWND hwndFrame;
QMSG qmsg;
HMQ hmq;

while( WinGetMsg( hab, &qmsg, NULL, 0, 0 ) )
    WinDispatchMsg( hab, /* PM anchor block handle */
                   &qmsg ); /* pointer to message */

/* Destroy the standard windows if they were created. */

if ( hwndFrame != NULL )
    WinDestroyWindow( hwndFrame ); /* frame window handle */

/* Destroy the message queue and release the anchor block. */

if ( hmq != NULL )
    WinDestroyMsgQueue( hmq );

if ( hab != NULL )
    WinTerminate( hab );
```

WinTerminate - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinTerminateApp

WinTerminateApp - Syntax

This function terminates an application previously started with the [WinStartApp](#) function.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAPP    happ; /* Application handle. */
BOOL    rc;    /* Termination indicator. */

rc = WinTerminateApp(happ);
```

WinTerminateApp Parameter - happ

happ ([HAPP](#)) - input
Application handle.

Identifies the application to terminate.

WinTerminateApp Return Value - rc

rc ([BOOL](#)) - returns
Termination indicator.

TRUE

Application successfully terminated
NULL
Error occurred.

WinTerminateApp - Parameters

happ ([HAPP](#)) - input
Application handle.

Identifies the application to terminate.

rc ([BOOL](#)) - returns
Termination indicator.

TRUE	Application successfully terminated
NULL	Error occurred.

WinTerminateApp - Remarks

The application to terminate must have been started using the [WinStartApp](#) function with the SAF_STARTCHILDAPP option specified.

If the specified application does not stop, this function returns TRUE. To ensure that the application has terminated, the application calling WinTerminateApp must wait for the appropriate message to be posted to the window specified in the [WinStartApp](#) function.

The WinTerminateApp function must be called from the same process as the [WinStartApp](#) function.

This function requires the existence of a message queue.

WinTerminateApp - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HAPP (0x1533)
The application handle passed to WinTerminateApp does not correspond to a valid session.

PMERR_CANNOT_STOP (0x1534)
The session cannot be stopped.

WinTerminateApp - Related Functions

Related Functions

- [WinStartApp](#)
-

WinTerminateApp - Example Code

This example calls WinTerminate in a typical termination sequence.

```
#define INCL_DOSSESMGR
#include <os2.h>

HWND      hwndNotify;
PPROGDETAILS pDetails;
HAPP      happ;

pDetails->Length          = sizeof(PROGDETAILS);
pDetails->progt.progc      = PROG_WINDOWABLEVIO;
pDetails->progt.fbVisible  = SHE_VISIBLE;
pDetails->pszTitle         = "TEXT";
pDetails->pszExecutable    = "TEXT.EXE";
pDetails->pszParameters    = NULL;
pDetails->pszStartupDir    = "";
pDetails->pszICON          = "T.ICO";
pDetails->pszEnvironment   = "WORKPLACE\\0\\0";
pDetails->swpInitial.fl    = SWP_ACTIVATE; /* Window      */
                                           /* positioning */
pDetails->swpInitial.cy    = 0;             /* Width of window */
pDetails->swpInitial.cx    = 0;             /* Height of window */
pDetails->swpInitial.y      = 0;             /* Lower edge of window */
pDetails->swpInitial.x      = 0;             /* Left edge of window */
pDetails->swpInitial.hwndInsertBehind = HWND_TOP;
pDetails->swpInitial.hwnd  = hwndNotify;
pDetails->swpInitial.ulReserved1 = 0;
pDetails->swpInitial.ulReserved2 = 0;

happ = WinStartApp(hwndNotify,
                   pDetails,
                   NULL, NULL,
                   SAF_STARTCHILDAPP);
.
.
.
WinTerminateApp(happ);
```

WinTerminateApp - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinTrackRect

WinTrackRect - Syntax

This function draws a tracking rectangle.

```
#define INCL_WINTRACKRECT /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwnd;          /* Window handle where tracking is to take place. */
HPS       hps;           /* Presentation-space handle. */
PTRACKINFO ptiTrackinfo; /* Track information. */
BOOL      rc;            /* Success indicator. */

rc = WinTrackRect(hwnd, hps, ptiTrackinfo);
```

WinTrackRect Parameter - hwnd

hwnd (**HWND**) - input

Window handle where tracking is to take place.

It is assumed that the style of this window is not WS_CLIPCHILDREN.

HWND_DESKTOP

Track over the entire screen

Other

Track over specified window only.

WinTrackRect Parameter - hps

hps (**HPS**) - input

Presentation-space handle.

Used for drawing the clipping rectangle:

NULLHANDLE

The *hwnd* parameter is used to calculate a presentation space for tracking. It is assumed that tracking takes place within *hwnd* and that the style of this window is not WS_CLIPCHILDREN. Thus, when the drag rectangle appears, it is not clipped by any children within the window. If the window style is WS_CLIPCHILDREN and the application causes the drag rectangle to be clipped, it must explicitly pass an appropriate presentation space.

Other

Specified presentation-space handle.

WinTrackRect Parameter - ptiTrackinfo

ptiTrackinfo (**PTRACKINFO**) - in/out

Track information.

WinTrackRect Return Value - rc

rc (**BOOL**) - returns
Success indicator.

TRUE

Tracking successful.

FALSE

Tracking canceled, or the pointing device was already captured when this function was called.

Only one tracking rectangle can be in use at one time.

WinTrackRect - Parameters

hwnd (**HWND**) - input
Window handle where tracking is to take place.

It is assumed that the style of this window is not WS_CLIPCHILDREN.

HWND_DESKTOP

Track over the entire screen

Other

Track over specified window only.

hps (**HPS**) - input
Presentation-space handle.

Used for drawing the clipping rectangle:

NULLHANDLE

The *hwnd* parameter is used to calculate a presentation space for tracking. It is assumed that tracking takes place within *hwnd* and that the style of this window is not WS_CLIPCHILDREN. Thus, when the drag rectangle appears, it is not clipped by any children within the window. If the window style is WS_CLIPCHILDREN and the application causes the drag rectangle to be clipped, it must explicitly pass an appropriate presentation space.

Other

Specified presentation-space handle.

ptiTrackinfo (**PTRACKINFO**) - in/out
Track information.

rc (**BOOL**) - returns
Success indicator.

TRUE

Tracking successful.

FALSE

Tracking canceled, or the pointing device was already captured when this function was called.

Only one tracking rectangle can be in use at one time.

WinTrackRect - Remarks

The WinTrackRect call provides general-purpose pointing-device tracking. It draws a rectangle and enables the user to position the entire rectangle, or size a specific side or corner, as required. The resulting rectangle is then returned to the application, which can use this new information for size and position data. The window manager interface for moving and sizing windows by means of the wide sizing borders uses this function, for example.

This function enables the caller to control such limiting values as:

- A maximum and minimum tracking size
- Absolute tracking-position limits
- The tracking rectangle side widths
- A restriction of tracking rectangle movements to a predefined positional grid.

It automatically calls [WinLockWindowUpdate](#) to prevent output in the window *hwnd* and its descendants while tracking. When tracking has been completed, output is enabled before this function returns. It also determines which button of the pointing device is depressed at the start of the operation, and only completes the tracking operation when the same button is released.

If the parameter of the [TRACKINFO](#) structure specified by the value is included, the pointing device pointer is positioned at the center of the tracking rectangle. Otherwise, the pointing device pointer is not moved from its current position and *delta* is established between the pointing device position and the part of the tracking rectangle that it moves (the *delta* is kept constant).

While moving or sizing with the keyboard interface, the pointing device pointer is repositioned with the tracking rectangle's new size or position.

While tracking, these keys are active:

Enter	Accepts the new position or size.
Left cursor	Moves the pointing device pointer and tracking rectangle left. If the pointing device pointer is on the upper or lower edge of the tracking rectangle, the pointer is moved to the top-left or bottom-left corner respectively.
Up cursor	Moves the pointing device pointer and tracking rectangle up. If the pointing device pointer is on the left or right edge of the tracking rectangle, the pointer is moved to the top-left or top-right corner respectively.
Right cursor	Moves the pointing device pointer and tracking rectangle right. If the pointing device pointer is on the upper or lower edge of the tracking rectangle, the pointer is moved to the top-right or bottom-right corner respectively.
Down cursor	Moves the pointing device pointer and tracking rectangle down. If the pointing device pointer is on the left or right edge of the tracking rectangle, the pointer is moved to the bottom-left or bottom-right corner respectively.
Esc	Cancels the current tracking operation. The value of the tracking rectangle is undefined on exit.

The pointing device and the keyboard interface can be intermixed. The caller need not include the value to use the keyboard interface, as this value simply initializes the position of the pointing device pointer.

If is specified in the [TRACKINFO](#) structure, the interior of the tracking rectangle is restricted to multiples of the values of the parameters. The default values for these are the system font character width and half the system font character height, respectively.

Tracking movements using the keyboard arrow keys depend on whether or not is specified in the [TRACKINFO](#) structure. If not specified, the increments are the values of the keyboard arrow keys do not cause tracking.

The tracking rectangle is usually logically "on top" of objects it tracks, so that the user can see the old size and position while tracking the new. Thus, it is possible for a window "below" the tracking rectangle to be updated while part of the tracking rectangle is "above" it.

Because the tracking rectangle is drawn in exclusive-OR mode, no window can draw below the tracking rectangle (and thereby obliterate it) without first notifying the tracking code, because unwanted areas of the tracking rectangle can be left behind. If the window doing the drawing is clipped out from the window in which the tracking is occurring, this problem does not arise.

To prevent a window that is currently processing a [WM_PAINT](#) message drawing over the tracking rectangle, the tracking rectangle is considered as a system-wide resource, only one of which can be in use at any time. If there is a risk of the currently-updating window

drawing on the tracking rectangle, the tracking rectangle is removed while that window and its child windows update, and it is then replaced. This is done during the [WinBeginPaint](#) and [WinEndPaint](#) functions. If the tracking rectangle overlaps, it is removed in the [WinBeginPaint](#) function. In the [WinEndPaint](#) function, all the child windows are updated by means of the [WinUpdateWindow](#) function before the tracking rectangle is redrawn.

WinTrackRect has a modal loop within it. The loop has a HK_MSGFILTER hook and a MSGF_TRACK hook code.

Note: The rectangle tracked by this function stays within the specified tracking bounds and dimensions. If the rectangle passed is out of these bounds, or it is too large or too small, it is modified to a rectangle that meets these limits.

WinTrackRect - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinTrackRect - Related Functions

Related Functions

- [WinShowTrackRect](#)
- [WinTrackRect](#)

WinTrackRect - Related Messages

Related Messages

- [WM_PAINT](#)

WinTrackRect - Example Code

This example shows how WinTrackRect can be used to allow a user size a rectangle on the screen.

```
#define INCL_WINTRACKRECT

#include <os2.h>

BOOL MyTrackRoutine(HAB hab, HPS hps, PRECTL rcl)
{
    TRACKINFO track;

    track.cxBorder = 4;
    track.cyBorder = 4; /* 4 pel wide lines used for rectangle */
    track.cxGrid = 1;
    track.cyGrid = 1; /* smooth tracking with mouse */
    track.cxKeyboard = 8;
    track.cyKeyboard = 8; /* faster tracking using cursor keys */
```

```

WinCopyRect(hab, &track.rclTrack, rcl);    /* starting point */

WinSetRect(hab, &track.rclBoundary, 0, 0, 640, 480); /* bounding rectangle */

track.ptlMinTrackSize.x = 10;
track.ptlMinTrackSize.y = 10; /* set smallest allowed size of rectangle */
track.ptlMaxTrackSize.x = 200;
track.ptlMaxTrackSize.y = 200; /* set largest allowed size of rectangle */

track.fs = TF_MOVE;

if (WinTrackRect(HWND_DESKTOP, hps, &track) )
{
    /* if successful copy final position back */
    WinCopyRect(hab, rcl, &track.rclTrack);
    return(TRUE);
}
else
{
    return(FALSE);
}
}

```

WinTrackRect - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinTranslateAccel

WinTranslateAccel - Syntax

This function translates a [WM_CHAR](#) message.

```

#define INCL_WINACCELERATORS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;           /* Anchor-block handle. */
HWND     hwnd;          /* Destination window. */
HACCEL   haccelAccel;   /* Accelerator-table handle. */
PQMSG    pQmsg;         /* Message to be translated. */
BOOL     rc;            /* Success indicator. */

rc = WinTranslateAccel(hab, hwnd, haccelAccel,
    pQmsg);

```

WinTranslateAccel Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinTranslateAccel Parameter - hwnd

hwnd ([HWND](#)) - input
Destination window.

WinTranslateAccel Parameter - haccelAccel

haccelAccel ([HACCEL](#)) - input
Accelerator-table handle.

WinTranslateAccel Parameter - pQmsg

pQmsg ([PQMSG](#)) - in/out
Message to be translated.

WinTranslateAccel Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinTranslateAccel - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

hwnd ([HWND](#)) - input
Destination window.

hacclAccel ([HACCEL](#)) - input
Accelerator-table handle.

pQmsg ([PQMSG](#)) - in/out
Message to be translated.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinTranslateAccel - Remarks

This function translates *pQmsg* if it is a [WM_CHAR](#) message in the accelerator table *hacclAccel*. The message is translated into a [WM_COMMAND](#), [WM_SYSCOMMAND](#), or [WM_HELP](#) message, with *hwnd* identifying the destination window. Normally, this parameter is a frame-window handle. This function does not highlight menu items.

If *hacclAccel* equals NULL, the current accelerator table is assumed.

WinTranslateAccel returns TRUE if the message matches an accelerator in the table. *pQmsg* is modified by WinTranslateAccel if a match is found.

If a menu item exists that matches the accelerator-command value, and that item is disabled, *pQmsg* is translated to a [WM_NULL](#) message, rather than a [WM_COMMAND](#), [WM_SYSCOMMAND](#), or [WM_HELP](#) message. If the command is [WM_SYSCOMMAND](#) or [WM_HELP](#) (and if a [WM_SYSCOMMAND](#) or FID_SYSMENU child window is searched) the menu child window of *hwnd* that has the FID_MENU identifier is searched.

It is possible to have accelerators that do not correspond to items in a menu. If the command value does not match any items in the menu, the message is still translated.

Generally, applications do not have to call this function; it is usually called automatically by [WinGetMsg](#) and [WinPeekMsg](#), when a [WM_CHAR](#) message is received with the window handle of the active window as the first parameter. The standard frame window procedure always passes [WM_COMMAND](#) messages to the FID_CLIENT window. Because the message is physically changed by WinTranslateAccel, applications do not see the [WM_CHAR](#) messages that result in [WM_COMMAND](#), [WM_SYSCOMMAND](#), or [WM_HELP](#) messages.

WinTranslateAccel - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_INVALID_HACCEL (0x101A)
An invalid accelerator-table handle was specified.

WinTranslateAccel - Related Functions

Related Functions

- [WinCopyAccelTable](#)
 - [WinCreateAccelTable](#)
 - [WinDestroyAccelTable](#)
 - [WinLoadAccelTable](#)
 - [WinQueryAccelTable](#)
 - [WinSetAccelTable](#)
 - [WinTranslateAccel](#)
-

WinTranslateAccel - Related Messages

Related Messages

- [WM_CHAR](#)
 - [WM_COMMAND](#)
 - [WM_HELP](#)
 - [WM_NULL](#)
 - [WM_SYSCOMMAND](#)
-

WinTranslateAccel - Example Code

This example uses the WinTranslateAccel API to translate WM_CHAR messages destined for the frame window.

```
#define INCL_WINWINDOWMGR
#define INCL_WINACCELERATORS
#include <OS2.H>

HACCEL haccel;
HWND hwndFrame, hwndClient; /* window handles. */
HAB hab; /* anchor block. */
QMSG qmsg;

hwndFrame = WinQueryWindow(hwndClient,
                             QW_PARENT); /* get handle of parent, */
                                           /* which is frame window. */

/* Now get the accel table for the frame window */
haccel = WinQueryAccelTable(hab,
                             hwndFrame);

WinTranslateAccel(hab,
                  hwndFrame,
                  haccel,
                  &qmsg);

switch(qmsg.msg)
{
    case WM_COMMAND:

    case WM_SYSCOMMAND:

    case WM_HELP:
        break;
}
```

WinTranslateAccel - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinUnionRect

WinUnionRect - Syntax

This function calculates a rectangle that bounds the two source rectangles.

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
PRECTL   prclDest;     /* Bounding rectangle. */
PRECTL   prclSrc1;     /* First source rectangle. */
PRECTL   prclSrc2;     /* Second source rectangle. */
BOOL      rc;          /* Nonempty indicator. */

rc = WinUnionRect(hab, prclDest, prclSrc1,
                  prclSrc2);
```

WinUnionRect Parameter - hab

hab ([HAB](#)) - input
Anchor-block handle.

WinUnionRect Parameter - prclDest

prclDest ([PRECTL](#)) - output
Bounding rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) may also be used, if supported by the language.

WinUnionRect Parameter - prclSrc1

prclSrc1 ([PRECTL](#)) - input
First source rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) may also be used, if supported by the language.

WinUnionRect Parameter - prclSrc2

prclSrc2 ([PRECTL](#)) - input
Second source rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) may also be used, if supported by the language.

WinUnionRect Return Value - rc

rc ([BOOL](#)) - returns
Nonempty indicator.

TRUE

prclDest is a nonempty rectangle

FALSE

Error, or *prclDest* is an empty rectangle.

WinUnionRect - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

prclDest ([PRECTL](#)) - output
Bounding rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) may also be used, if supported by the language.

prclSrc1 ([PRECTL](#)) - input
First source rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) may also be used, if supported by the language.

prclSrc2 ([PRECTL](#)) - input
Second source rectangle.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) may also be used, if supported by the language.

rc ([BOOL](#)) - returns
Nonempty indicator.

TRUE

prclDest is a nonempty rectangle

FALSE

Error, or *prclDest* is an empty rectangle.

WinUnionRect - Remarks

prclSrc1 and *prclSrc2* must not be NULL pointers, although the rectangles they point to can be empty (see the [WinIsRectEmpty](#) function).

If one of the source rectangles is empty, the other is returned.

WinUnionRect - Related Functions

Related Functions

- [WinCopyRect](#)
- [WinEqualRect](#)
- [WinFillRect](#)
- [WinInflateRect](#)
- [WinIntersectRect](#)
- [WinIsRectEmpty](#)
- [WinOffsetRect](#)
- [WinPtInRect](#)
- [WinSetRect](#)
- [WinSetRectEmpty](#)
- [WinSubtractRect](#)
- [WinUnionRect](#)

WinUnionRect - Example Code

This example uses the WinUnionRect call to find a rectangle that bounds two source rectangles.

```
#define INCL_WINRECTANGLES
#include <OS2.H>
HAB hab;
RECTL resultrcl;    /* result. */
RECTL rcla={25,     /* x coordinate of left-hand edge of */
```

```

        /* rectangle. */
    25,    /* y coordinate of bottom edge of    */
        /* rectangle. */
    125,   /* x coordinate of right-hand edge of */
        /* rectangle. */
    125};  /* y coordinate of top edge of rectangle. */

RECTL rclb = {15, /* x coordinate of left-hand edge of */
              /* rectangle. */
              15, /* y coordinate of bottom edge of    */
              /* rectangle. */
              125, /* x coordinate of right-hand edge of */
              /* rectangle. */
              125};

WinUnionRect(hab,
             &resultrcl,
             &rcla,
             &rclb);

```

WinUnionRect - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinUnlockSystem

WinUnlockSystem - Syntax

This function is specific to OS/2 Version 2.1 or higher.

This function causes an application program to attempt to unlock the system.

```

#define INCL_WINMESSAGEGR
#include <os2.h>

HAB      hab;           /* The application anchor block. */
PSZ      pszPassword;   /* Password string. */
BOOL     rc;            /* Return value. */

rc = WinUnlockSystem(hab, pszPassword);

```

WinUnlockSystem Parameter - hab

hab (**HAB**) - input
The application anchor block.

WinUnlockSystem Parameter - pszPassword

pszPassword (**PSZ**) - input
Password string.

WinUnlockSystem Return Value - rc

rc (**BOOL**) - returns
Return value.

TRUE

The system was successfully unlocked.

FALSE

An error occurred or the system was already in an unlocked state.

WinUnlockSystem - Parameters

hab (**HAB**) - input
The application anchor block.

pszPassword (**PSZ**) - input
Password string.

rc (**BOOL**) - returns
Return value.

TRUE

The system was successfully unlocked.

FALSE

An error occurred or the system was already in an unlocked state.

WinUnlockSystem - Remarks

This function causes an application program to attempt to unlock the system by passing a text string that can be compared against the system lockup password that is stored in an encrypted form by the system.

If the string passed in matches the system lockup password, the lockup dialog is dismissed and the user is able to use his machine. Otherwise, the system remains in the locked up state.

OS/2 does not provide a method for querying *pszPassword* to find out the password. It is up to the developer to ask the user what the password is.

In order to execute WinUnlockSystem after a WinLockupSystem has been called, the WinUnlockSystem must be called from a separate thread because WinLockupSystem will not return until a password has been entered from the keyboard.

The LockupHook hook allows a PM application to customize system lockups.

In order for the window to appear as the top most window on the lockup screen, the WS_CLIPSIBLINGS flag must be used for the *fStyle* parameter in the WinCreateWindow or WinCreateStdWindow call.

WinUnlockSystem - Related Functions

Related Functions

- [LockupHook](#)
- [WinLockupSystem](#)

WinUnlockSystem - Example Code

```
HAB    hab          = NULLHANDLE;
HMQ    hmq          = NULLHANDLE;
QMSG   qmsg         = {0};
HWND   hwndFrame    = NULLHANDLE;
HWND   hwndClient   = NULLHANDLE;
CHAR   *szAppName   = "Lockup  ";
ULONG  FrameFlags   = FCF_STANDARD;
PSZ    pszPassword   = "m4a3x28t";

hab = WinInitialize( 0 );
hmq = WinCreateMsgQueue( hab, 0 );

WinRegisterClass( hab, szAppName, ClientWndProc,
                  CS_CLIPSIBLINGS | CS_CLIPCHILDREN | CS_SIZEREDRAW, 0 );

hwndFrame = WinCreateStdWindow( HWND_DESKTOP, WS_VISIBLE, &FrameFlags,
                                szAppName, szAppName, 0L,
                                NULLHANDLE, ID_PLATFORM, &hwndClient );

WinLockupSystem( hab );      /* Don't allow user to interact with
                              application unless password is known */

/* Application code here ... */

WinUnlockSystem( hab, pszPassword );  /* Unlock system when ready */
.
.
.
```

WinUnlockSystem - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinUpdateWindow

WinUpdateWindow - Syntax

This function forces the update of a window and its associated child windows.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd; /* Window handle. */
BOOL    rc;    /* Window-updated indicator. */

rc = WinUpdateWindow(hwnd);
```

WinUpdateWindow Parameter - hwnd

hwnd (**HWND**) - input
Window handle.

WinUpdateWindow Return Value - rc

rc (**BOOL**) - returns
Window-updated indicator.

TRUE	Window successfully updated
FALSE	Window not successfully updated.

WinUpdateWindow - Parameters

hwnd ([HWND](#)) - input
Window handle.

rc ([BOOL](#)) - returns
Window-updated indicator.

TRUE	Window successfully updated
FALSE	Window not successfully updated.

WinUpdateWindow - Remarks

If *hwnd* is an asynchronous window (that is, it does not have a style of [WS_SYNCPAINT](#)), only it and its asynchronous children are updated. They are sent [WM_PAINT](#) messages from this function. If the window is owned by a different thread from the thread issuing the call, the message is sent asynchronously (using [WinPostMsg](#)) and not synchronously (using [WinSendMessage](#)).

If the window does not have an invalid region, a [WM_PAINT](#) message might not be sent.

If *hwnd* is a child of a nonclip-children parent, the update region of *hwnd* is subtracted from the update region of the parent, if the parent has one. This is so that any parent-window drawing after *hwnd* does not draw over whatever is drawn by *hwnd*.

WinUpdateWindow - Errors

Possible returns from [WinGetLastError](#)

[PMERR_INVALID_HWND](#) (0x1001)
An invalid window handle was specified.

WinUpdateWindow - Related Functions

Related Functions

- [WinBeginPaint](#)
- [WinEnableWindowUpdate](#)
- [WinEndPaint](#)
- [WinExcludeUpdateRegion](#)
- [WinGetClipPS](#)
- [WinGetPS](#)
- [WinGetScreenPS](#)
- [WinInvalidateRect](#)
- [WinInvalidateRegion](#)
- [WinIsWindowShowing](#)
- [WinIsWindowVisible](#)
- [WinLockVisRegions](#)
- [WinOpenWindowDC](#)
- [WinQueryUpdateRect](#)
- [WinQueryUpdateRegion](#)
- [WinRealizePalette](#)
- [WinReleasePS](#)
- [WinShowWindow](#)
- [WinUpdateWindow](#)
- [WinValidateRect](#)
- [WinValidateRegion](#)

WinUpdateWindow - Related Messages

Related Messages

- [WM_PAINT](#)
-

WinUpdateWindow - Example Code

This example uses the WinUpdateWindow call to send a WM_PAINT message to a window procedure.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
#define WM_USERDEF WM_USER + 1
main()
{
}

static MRESULT ClientWindowProc( HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2 )
{
    switch(msg)
    {
        case WM_PAINT:
            break;
        case WM_USERDEF:
            WinUpdateWindow(hwnd);
    }
}
```

WinUpdateWindow - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinUpper

WinUpper - Syntax

This function converts a string to uppercase.

```
#define INCL_WINCOUNTRY /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
ULONG    ulCodepage;   /* Code page. */
ULONG    ulCountry;    /* Country code. */
PSZ      pszString;    /* String to be converted to uppercase. */
ULONG    ulRetLen;     /* Length of converted string. */

ulRetLen = WinUpper(hab, ulCodepage, ulCountry,
                    pszString);
```

WinUpper Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinUpper Parameter - ulCodepage

ulCodepage (**ULONG**) - input
Code page.

- 0
Use the current-process code page
- Other
Use the specified code page.

WinUpper Parameter - ulCountry

ulCountry (**ULONG**) - input
Country code.

- 0
Use the default country code specified in CONFIG.SYS
- Other
Use the specified country code.

WinUpper Parameter - pszString

pszString (PSZ) - in/out
String to be converted to uppercase.

WinUpper Return Value - ulRetLen

ulRetLen (ULONG) - returns
Length of converted string.

WinUpper - Parameters

hab (HAB) - input
Anchor-block handle.

ulCodepage (ULONG) - input
Code page.

- 0 Use the current-process code page
- Other Use the specified code page.

ulCountry (ULONG) - input
Country code.

- 0 Use the default country code specified in CONFIG.SYS
- Other Use the specified country code.

pszString (PSZ) - in/out
String to be converted to uppercase.

ulRetLen (ULONG) - returns
Length of converted string.

WinUpper - Remarks

The following is the list of valid code pages:

Country	Code Page
Canadian-French	863
Desktop Publishing	1004
Iceland	861
Latin 1 Multilingual	850
Latin 2 Multilingual	852
Nordic	865
Portuguese	860
Turkey	857
U.S. (IBM PC)	437

Code page 1004 is compatible with Microsoft** Windows**.

The following EBCDIC code pages, based on character set 697, are also available for output:

Country	Code Page
Austrian/German	273
Belgian	500
Brazil	037
Czechoslovakia	870
Danish/Norwegian	277
Finnish/Swedish	278
French	297
Hungary	870
Iceland	871
International	500
Italian	280
Poland	870
Portuguese	037
Spanish	284
Turkey	1026
U.K.-English	285
U.S.-English	037
Yugoslavia	870

Code pages 274 (Belgian) and 282 (Portuguese) can be used to provide access to old data. The following is the list of valid country codes:

Country	Code
Arabic	785
Australian	61
Belgian	32
Canadian-French	2
Danish	45
Finnish	358
French	33
German	49
Hebrew	972
Italian	39
Japanese	81
Korean	82
Latin-American	3
Netherlands	31
Norwegian	47
Portuguese	351
Simpl. Chinese	86
Spanish	34
Swedish	46
Swiss	41
Trad. Chinese	88
UK-English	44
US-English	1
Other country	0

WinUpper - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_STRING_PARM (0x100B)
The specified string parameter is invalid.

WinUpper - Related Functions

Related Functions

- [WinCompareStrings](#)
- [WinLoadString](#)
- [WinNextChar](#)
- [WinPrevChar](#)
- [WinSubstituteStrings](#)
- [WinUpper](#)
- [WinUpperChar](#)

WinUpper - Example Code

This example shows how the WinUpper call can be used to convert a strings in NLS languages to uppercase.

```
#define INCL_WINCOUNTRY
#include <OS2.H>
#include <stdio.h>
main()
{
    HAB hab;
    char szString[] = "hablas español?";
    hab = WinInitialize(0);
    WinUpper(hab,
             850,
             34,
             szString);
    WinTerminate(hab);
}
```

WinUpper - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinUpperChar

WinUpperChar - Syntax

This function translates a character to uppercase.

```

#define INCL_WINCOUNTRY /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
ULONG    ulCodepage;   /* Code page. */
ULONG    ulCountry;    /* Country code. */
ULONG    ulInchar;     /* Character to be translated to uppercase. */
ULONG    ulOutchar;    /* Translated character. */

ulOutchar = WinUpperChar(hab, ulCodepage,
                        ulCountry, ulInchar);

```

WinUpperChar Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinUpperChar Parameter - ulCodepage

ulCodepage (**ULONG**) - input
Code page.

- 0 Use the current-process code page
- Other Use the specified code page.

WinUpperChar Parameter - ulCountry

ulCountry (**ULONG**) - input
Country code.

- 0 Use the default country code specified in CONFIG.SYS
- Other Use the specified country code.

WinUpperChar Parameter - ulInchar

ulInchar (**ULONG**) - input
Character to be translated to uppercase.

WinUpperChar Return Value - ulOutchar

ulOutchar ([ULONG](#)) - returns
Translated character.

0	Error occurred
Other	The translated character.

WinUpperChar - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

ulCodepage ([ULONG](#)) - input
Code page.

0	Use the current-process code page
Other	Use the specified code page.

ulCountry ([ULONG](#)) - input
Country code.

0	Use the default country code specified in CONFIG.SYS
Other	Use the specified country code.

ullnchar ([ULONG](#)) - input
Character to be translated to uppercase.

ulOutchar ([ULONG](#)) - returns
Translated character.

0	Error occurred
Other	The translated character.

WinUpperChar - Remarks

The case-mapping used is the same as provided by the OS/2 DosCaseMap call.

WinUpperChar - Errors

Possible returns from [WinGetLastError](#)

PMERR_INV_CODEPAGE (0x2052)
An invalid code-page was specified.

PMERR_INVALID_STRING_PARM (0x100B)
The specified string parameter is invalid.

WinUpperChar - Related Functions

Related Functions

- [WinCompareStrings](#)
- [WinLoadString](#)
- [WinNextChar](#)
- [WinPrevChar](#)
- [WinSubstituteStrings](#)
- [WinUpper](#)
- WinUpperChar

WinUpperChar - Example Code

This example shows how the WinUpperChar call can be used to convert a characters in NLS languages to uppercase.

```
#define INCL_WINCOUNTRY
#include <OS2.H>
#include <stdio.h>

main()
{
    HAB hab;
    char szString[] = "ö";
    hab = WinInitialize(0);

    WinUpper(hab,
             850,
             49,
             szString);
    WinTerminate(hab);
}
```

WinUpperChar - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinValidateRect

WinValidateRect - Syntax

This function subtracts a rectangle from the update region of an asynchronous paint window, marking that part of the window as visually valid.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwnd;          /* Handle of window whose update region is changed. */
PRECTL    prclRect;      /* Rectangle to be subtracted from the window's update region. */
BOOL      fIncludeClippedChildren; /* Validation-scope indicator. */
BOOL      rc;            /* Success indicator. */

rc = WinValidateRect(hwnd, prclRect, fIncludeClippedChildren);
```

WinValidateRect Parameter - hwnd

hwnd (**HWND**) - input

Handle of window whose update region is changed.

If this parameter is **HWND_DESKTOP** or a desktop-window handle, the function applies to the whole screen (or desktop).

WinValidateRect Parameter - prclRect

prclRect (**PRECTL**) - input

Rectangle to be subtracted from the window's update region.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type **WRECT** may also be used, if supported by the language.

WinValidateRect Parameter - fIncludeClippedChildren

fIncludeClippedChildren (**BOOL**) - input

Validation-scope indicator.

TRUE	Include descendants of <i>hwnd</i> in the valid rectangle
FALSE	Include descendants of <i>hwnd</i> in the valid rectangle, only if parent is not WS_CLIPCHILDREN.

WinValidateRect Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinValidateRect - Parameters

hwnd ([HWND](#)) - input
Handle of window whose update region is changed.

If this parameter is HWND_DESKTOP or a desktop-window handle, the function applies to the whole screen (or desktop).

prclRect ([PRECTL](#)) - input
Rectangle to be subtracted from the window's update region.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type [WRECT](#) may also be used, if supported by the language.

fIncludeClippedChildren ([BOOL](#)) - input
Validation-scope indicator.

TRUE	Include descendants of <i>hwnd</i> in the valid rectangle
FALSE	Include descendants of <i>hwnd</i> in the valid rectangle, only if parent is not WS_CLIPCHILDREN.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinValidateRect - Remarks

The call is not used for CS_SYNCPAINT windows.

This function has no effect on the window if any part of the window has been made invalid since the last call to [WinBeginPaint](#), [WinQueryUpdateRect](#), or [WinQueryUpdateRegion](#).

WinValidateRect - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

PMERR_INVALID_FLAG (0x1019)
An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

WinValidateRect - Related Functions

Related Functions

- [WinBeginPaint](#)
 - [WinEnableWindowUpdate](#)
 - [WinEndPaint](#)
 - [WinExcludeUpdateRegion](#)
 - [WinGetClipPS](#)
 - [WinGetPS](#)
 - [WinGetScreenPS](#)
 - [WinInvalidateRect](#)
 - [WinInvalidateRegion](#)
 - [WinIsWindowShowing](#)
 - [WinIsWindowVisible](#)
 - [WinLockVisRegions](#)
 - [WinOpenWindowDC](#)
 - [WinQueryUpdateRect](#)
 - [WinQueryUpdateRegion](#)
 - [WinRealizePalette](#)
 - [WinReleasePS](#)
 - [WinShowWindow](#)
 - [WinUpdateWindow](#)
 - [WinValidateRect](#)
 - [WinValidateRegion](#)
-

WinValidateRect - Example Code

The window needs painting. This is done asynchronously on the drawing thread. The window update region is copied into a local region and passed to the drawing thread. The window must be validated now (to prevent further unnecessary paint messages).

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
HRGN hrgnUpdate;
HPS hps;
HWND hwnd;
/* Window needs paint */
case WM_PAINT:

/* assume we stop any asynchronous drawing. */
/* by posting a message to the asynchronous */
/* drawing thread. */

hrgnUpdate=(HRGN)GpiCreateRegion(hps, /* Create empty region */
                                0L,
                                (PRECTL)NULL);
```

```

WinQueryUpdateRegion(hwnd,          /* Save the window update */
                    hrgnUpdate);    /* region.                */

WinValidateRect(hwnd,              /* Validate window now to */
                (PRECTL)NULL,      /* stop more paint msgs */
                TRUE);

/* assume a message is posted to the drawing thread, passing */
/* the update region: (MPARAM)hrgnUpdate.                    */

mr = (MRESULT) 0L;                /* Message processed      */
break;                            /* End window painting    */

```

WinValidateRect - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinValidateRegion

WinValidateRegion - Syntax

This function subtracts a region from the update region of an asynchronous paint window, marking that part of the window as visually valid.

```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwnd;                /* Handle of window whose update region is changed. */
HRGN    hrgn;                /* Handle of subtracted region. */
BOOL    fIncludeClippedChildren; /* Validation-scope indicator. */
BOOL    rc;                  /* Success indicator. */

rc = WinValidateRegion(hwnd, hrgn, fIncludeClippedChildren);

```

WinValidateRegion Parameter - hwnd

hwnd (HWND) - input
Handle of window whose update region is changed.

If this parameter is HWND_DESKTOP or a desktop window handle, the function applies to the whole screen (or desktop).

WinValidateRegion Parameter - hrgn

hrgn (HRGN) - input
Handle of subtracted region.

This is the region that is subtracted from the window's update region.

WinValidateRegion Parameter - fIncludeClippedChildren

fIncludeClippedChildren (BOOL) - input
Validation-scope indicator.

- | | |
|-------|--|
| TRUE | Include descendants of <i>hwnd</i> in the valid region |
| FALSE | Include descendants of <i>hwnd</i> in the valid region, only if parent is not WS_CLIPCHILDREN. |

WinValidateRegion Return Value - rc

rc (BOOL) - returns
Success indicator.

- | | |
|-------|-----------------------|
| TRUE | Successful completion |
| FALSE | Error occurred. |

WinValidateRegion - Parameters

hwnd (HWND) - input
Handle of window whose update region is changed.

If this parameter is HWND_DESKTOP or a desktop window handle, the function applies to the whole screen (or desktop).

hrgn (HRGN) - input
Handle of subtracted region.

This is the region that is subtracted from the window's update region.

flIncludeClippedChildren ([BOOL](#)) - input
Validation-scope indicator.

TRUE

Include descendants of *hwnd* in the valid region

FALSE

Include descendants of *hwnd* in the valid region, only if parent is not WS_CLIPCHILDREN.

rc ([BOOL](#)) - returns
Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinValidateRegion - Remarks

The call is not used for CS_SYNCPAINT windows.

The call has no effect on the window if any part of the window has been made invalid since the last call to [WinBeginPaint](#), [WinQueryUpdateRect](#), or [WinQueryUpdateRegion](#).

WinValidateRegion - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)

An invalid window handle was specified.

PMERR_HRGN_BUSY (0x2034)

An internal region busy error was detected. The region was locked by one thread during an attempt to access it from another thread.

PMERR_INVALID_FLAG (0x1019)

An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

WinValidateRegion - Related Functions

Related Functions

- [WinBeginPaint](#)
- [WinEnableWindowUpdate](#)
- [WinEndPaint](#)
- [WinExcludeUpdateRegion](#)
- [WinGetClipPS](#)
- [WinGetPS](#)
- [WinGetScreenPS](#)
- [WinInvalidateRect](#)
- [WinInvalidateRegion](#)
- [WinIsWindowShowing](#)
- [WinIsWindowVisible](#)
- [WinLockVisRegions](#)
- [WinOpenWindowDC](#)
- [WinQueryUpdateRect](#)
- [WinQueryUpdateRegion](#)

- [WinRealizePalette](#)
- [WinReleasePS](#)
- [WinShowWindow](#)
- [WinUpdateWindow](#)
- [WinValidateRect](#)
- [WinValidateRegion](#)

WinValidateRegion - Example Code

This example shows how an application can incrementally repaint an asynchronous-paint window one area at a time. While a window is invalid (has a non-null update region), WM_PAINT messages are returned by WinGetMsg. The application uses WinQueryUpdateRegion to obtain a region that requires repainting, and WinValidateRegion to validate the region (reset the update region to null).

```
#define INCL_WINWINDOWMGR
#define INCL_GPIREGIONS
#include <OS2.H>
HRGN hrgnUpdt, sRgnType;
HPS hpsPaint;
HWND hwnd;
/* Window needs paint */
case WM_PAINT:

/* assume we stop any asynchronous drawing. */
/* by posting a message to the asynchronous */
/* drawing thread. */

hrgnUpdt = (HRGN)GpiCreateRegion(hpsPaint,
                                (ULONG)0,
                                (PRECTL)NULL);

sRgnType = (HRGN)WinQueryUpdateRegion(hwnd,
                                       hrgnUpdt);

/* if the region is not null and the call is not in error, */
/* validate the region. */

if ((sRgnType != NULL) &&
    (sRgnType != RGN_ERROR)) {
    WinValidateRegion(hwnd, hrgnUpdt, FALSE);
/*
here we would send the update region handle to an
asynchronous drawing thread. We have already validated the
region, so no more WM_PAINT messages will be sent due to this
region.
*/
    } else { GpiDestroyRegion(hpsPaint, hrgnUpdt);}
```

WinValidateRegion - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Errors](#)
- [Remarks](#)
- [Example Code](#)
- [Related Functions](#)
- [Glossary](#)

WinWaitEventSem

WinWaitEventSem - Syntax

This function waits for an event semaphore to be posted for a Presentation Manager message to be received.

```
#define INCL_WINMESSAGEGR
#include <os2.h>

HEV      hev;          /* The handle of the event semaphore to wait for. */
ULONG    ulTimeout;    /* Time-out in milliseconds. */
APIRET   rc;           /* Return Code. */

rc = WinWaitEventSem(hev, ulTimeout);
```

WinWaitEventSem Parameter - hev

hev ([HEV](#)) - input
The handle of the event semaphore to wait for.

WinWaitEventSem Parameter - ulTimeout

ulTimeout ([ULONG](#)) - input
Time-out in milliseconds.

This is the maximum amount of time the user wants to allow the thread to be blocked.

This parameter can also have the following values:

SEM_IMMEDIATE_RETURN (0)
WinWaitMuxWaitSem returns without blocking the calling thread.
SEM_INDEFINITE_WAIT (-1)
WinWaitMuxWaitSem blocks the calling thread indefinitely.

WinWaitEventSem Return Value - rc

rc ([APIRET](#)) - returns
Return Code.

WinWaitEventSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
95	ERROR_INTERRUPT
640	ERROR_TIMEOUT

WinWaitEventSem - Parameters

hev ([HEV](#)) - input

The handle of the event semaphore to wait for.

ulTimeout ([ULONG](#)) - input

Time-out in milliseconds.

This is the maximum amount of time the user wants to allow the thread to be blocked.

This parameter can also have the following values:

SEM_IMMEDIATE_RETURN (0)

WinWaitMuxWaitSem returns without blocking the calling thread.

SEM_INDEFINITE_WAIT (-1)

WinWaitMuxWaitSem blocks the calling thread indefinitely.

rc ([APIRET](#)) - returns

Return Code.

WinWaitEventSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
95	ERROR_INTERRUPT
640	ERROR_TIMEOUT

WinWaitEventSem - Remarks

WinWaitEventSem is similar to DosWaitEventSem and enables a thread to wait for an event semaphore to be posted or for a window message sent by [WinSendMsg](#) from another thread to be received.

This function can be called by any thread in the process that created the semaphore. Threads in other processes can also call this function, but they must first gain access to the semaphore by calling DosOpenEventSem.

Since the processing of a window message may take longer than the value specified by the *ulTimeout* parameter, this function may not return within the time specified by that value.

If the main thread (which normally processes messages) is waiting on a semaphore for a prolonged period of time, messages will not be processed, and the "Bad App" dialog can occur. This is expected behavior. To avoid this, a message-box can be displayed telling the user that the window is being initialized. While the window loads, that message box can process the incoming system messages.

WinWaitEventSem - Related Functions

Related Functions

- [WinPostMsg](#)
- [WinSendMsg](#)

WinWaitEventSem - Example Code

This example causes the calling thread to wait until the specified event semaphore is posted. Assume that the handle of the semaphore has been placed into *hev* already.

ulTimeout is the number of milliseconds that the calling thread will wait for the event semaphore to be posted. If the specified event semaphore is not posted during this time interval, the request times out.

```
#define INCL_DOSSEMAPHORES /* Semaphore values */
#define INCL_WINMESSAGEGR
#include <os2.h>
#include <stdio.h>

#ifdef ERROR_TIMEOUT
#define ERROR_TIMEOUT 640
#define ERROR_INTERRUPT 95
#endif

HEV hev; /* Event semaphore handle */
ULONG ulTimeout; /* Number of milliseconds to wait */
ULONG rc; /* Return code */

ulTimeout = 60000; /* Wait for a maximum of 1 minute */

rc = WinWaitEventSem(hev, ulTimeout);

if (rc == ERROR_TIMEOUT)
{
    printf("WinWaitEventSem call timed out");
    return;
}

if (rc == ERROR_INTERRUPT)
{
    printf("WinWaitEventSem call was interrupted");
    return;
}

if (rc != 0)
{
    printf("WinWaitEventSem error: return code = %ld", rc);
    return;
}
```

WinWaitEventSem - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinWaitMsg

WinWaitMsg - Syntax

This function waits for a filtered message.

```
#define INCL_WINMESSAGEGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HAB      hab;          /* Anchor-block handle. */
ULONG    ulFirst;       /* First message identity. */
ULONG    ulLast;        /* Last message identity. */
BOOL     rc;            /* Success indicator. */

rc = WinWaitMsg(hab, ulFirst, ulLast);
```

WinWaitMsg Parameter - hab

hab (**HAB**) - input
Anchor-block handle.

WinWaitMsg Parameter - ulFirst

ulFirst (**ULONG**) - input
First message identity.

WinWaitMsg Parameter - ulLast

ulLast (**ULONG**) - input
Last message identity.

WinWaitMsg Return Value - rc

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinWaitMsg - Parameters

hab ([HAB](#)) - input
Anchor-block handle.

ulFirst ([ULONG](#)) - input
First message identity.

ulLast ([ULONG](#)) - input
Last message identity.

rc ([BOOL](#)) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinWaitMsg - Remarks

This function causes the current thread to wait for a message to arrive on the message queue associated with *hab*. This must be the next message since the queue was last inspected by a *rc* return from [WinGetMsg](#) or [WinPeekMsg](#). It must also conform to the filtering criteria specified by *ulFirst* and *ulLast*.

For details of the filtering performed by *ulFirst* and *ulLast*, see the [WinGetMsg](#) function.

WinWaitMsg - Related Functions

Related Functions

- [WinBroadcastMsg](#)
- [WinCreateMsgQueue](#)
- [WinDestroyMsgQueue](#)
- [WinDispatchMsg](#)
- [WinGetDlgMsg](#)
- [WinGetMsg](#)
- [WinInSendMessage](#)
- [WinPeekMsg](#)
- [WinPostMsg](#)
- [WinPostQueueMsg](#)
- [WinQueryMsgPos](#)
- [WinQueryMsgTime](#)
- [WinQueryQueueInfo](#)

- [WinQueryQueueStatus](#)
- [WinSendDlgItemMsg](#)
- [WinSendMsg](#)
- [WinSetClassMsgInterest](#)
- [WinSetMsgInterest](#)
- [WinSetMsgMode](#)
- [WinSetSynchroMode](#)
- [WinWaitMsg](#)

WinWaitMsg - Example Code

In this example the pointer is kept hidden until mouse activity is detected. The WinWaitMsg call is used to wait for any mouse message.

```
#define INCL_WINWINDOWMGR
#define INCL_WINPOINTERS
#define INCL_WINDESKTOP
#define INCL_WININPUT
#include <OS2.H>

HWND hwnd;
HPOINTER hpointer;
HAB hab;

/* Get the pointer handle */
hpointer = WinQueryPointer(HWND_DESKTOP);

/* Hide the mouse */
WinShowPointer(hwnd,FALSE);

/* All the mouse messages from WM_MOUSEFIRST */
/* to WM_BUTTON3DBLCLK inclusive */
WinWaitMsg(hab,
            WM_MOUSEFIRST,
            WM_BUTTON3DBLCLK);

/* If there has been any mouse activity, show the mouse */
WinShowPointer(hwnd,TRUE);
```

WinWaitMsg - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinWaitMuxWaitSem

WinWaitMuxWaitSem - Syntax

WinWaitMuxWaitSem waits for a muxwait semaphore to clear or for a Presentation Manager message.

```
#define INCL_WINMESSAGEGR
#include <os2.h>

HMUX      hmutex;      /* The handle of the muxwait semaphore to wait for. */
ULONG     ulTimeout;   /* Time-out in milliseconds. */
PULONG    pulUser;     /* Pointer to receive the user field. */
APIRET    ulrc;        /* Return Code. */

ulrc = WinWaitMuxWaitSem(hmutex, ulTimeout,
                        pulUser);
```

WinWaitMuxWaitSem Parameter - hmutex

hmutex (**HMUX**) - input
The handle of the muxwait semaphore to wait for.

WinWaitMuxWaitSem Parameter - ulTimeout

ulTimeout (**ULONG**) - input
Time-out in milliseconds.

This is the maximum amount of time the user wants to allow the thread to be blocked.

This parameter can also have the following values:

SEM_IMMEDIATE_RETURN (0)
WinWaitMuxWaitSem returns without blocking the calling thread.
SEM_INDEFINITE_WAIT (-1)
WinWaitMuxWaitSem blocks the calling thread indefinitely.

WinWaitMuxWaitSem Parameter - pulUser

pulUser (**PULONG**) - output
Pointer to receive the user field.

A pointer to receive the user field (from the muxwait semaphore data structure) of the semaphore that was posted or released.

If DCMW_WAIT_ANY was specified in the *flAttr* parameter when the muxwait semaphore was created, this will be the user field of the semaphore that was posted or released. If the muxwait semaphore consists of mutex semaphores, any mutex semaphore that is released is owned by the caller.

If DCMW_WAIT_ALL was specified in the *flAttr* parameter when the muxwait semaphore was created, this will be the user field of the *last* semaphore that was posted or released. (If the thread did not block, the last semaphore that was posted or released will also be the last semaphore in the muxwait-semaphore list.) If the muxwait semaphore consists of mutex semaphores, all of the mutex semaphores that are released are owned by the caller.

WinWaitMuxWaitSem Return Value - ulrc

ulrc ([APIRET](#)) - returns
Return Code.

WinWaitMuxWaitSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT
103	ERROR_TOO_MANY_SEM_REQUESTS
105	ERROR_SEM_OWNER_DIED
286	ERROR_EMPTY_MUXWAIT
287	ERROR_MUTEX_OWNED
292	ERROR_WRONG_TYPE
640	ERROR_TIMEOUT

WinWaitMuxWaitSem - Parameters

hmutex ([HMXU](#)) - input
The handle of the muxwait semaphore to wait for.

ulTimeout ([ULONG](#)) - input
Time-out in milliseconds.

This is the maximum amount of time the user wants to allow the thread to be blocked.

This parameter can also have the following values:

SEM_IMMEDIATE_RETURN (0)	WinWaitMuxWaitSem returns without blocking the calling thread.
SEM_INDEFINITE_WAIT (-1)	WinWaitMuxWaitSem blocks the calling thread indefinitely.

pulUser ([PULONG](#)) - output
Pointer to receive the user field.

A pointer to receive the user field (from the muxwait semaphore data structure) of the semaphore that was posted or released.

If DCMW_WAIT_ANY was specified in the *flAttr* parameter when the muxwait semaphore was created, this will be the user field of the semaphore that was posted or released. If the muxwait semaphore consists of mutex semaphores, any mutex semaphore that is released is owned by the caller.

If DCMW_WAIT_ALL was specified in the *flAttr* parameter when the muxwait semaphore was created, this will be the user field of the *last* semaphore that was posted or released. (If the thread did not block, the last semaphore that was posted or released will also be the last semaphore in the muxwait-semaphore list.) If the muxwait semaphore consists of mutex semaphores, all of the mutex semaphores that are released are owned by the caller.

ulrc ([APIRET](#)) - returns
Return Code.

WinWaitMuxWaitSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT
103	ERROR_TOO_MANY_SEM_REQUESTS
105	ERROR_SEM_OWNER_DIED
286	ERROR_EMPTY_MUXWAIT
287	ERROR_MUTEX_OWNED
292	ERROR_WRONG_TYPE
640	ERROR_TIMEOUT

WinWaitMuxWaitSem - Remarks

WinWaitMuxWaitSem is similar to DosWaitMuxWaitSem and enables a thread to wait for a muxwait semaphore to clear or for a window message sent by the WinSendMsg function from another thread to be received.

This function can be issued by any thread in the process that created the semaphore. Threads in other processes can also issue this function, but they must first gain access to the semaphore by issuing DosOpenMuxWaitSem.

Since the processing of a window message may take longer than the value specified by the *ulTimeout* parameter, this function may not return within the time specified by that value.

If the main thread (which normally processes messages) is waiting on a semaphore for a prolonged period of time, messages will not be processed, and the "Bad App" dialog can occur. This is expected behavior. To avoid this, a message-box can be displayed telling the user that the window is being initialized. While the window loads, that message box can process the incoming system messages.

WinWaitMuxWaitSem - Related Functions

Related Functions

- [WinPostMsg](#)
- [WinSendMsg](#)

WinWaitMuxWaitSem - Example Code

This example waits for a muxwait semaphore to clear. Assume that the handle of the semaphore has been placed into *hMux* already. *ulTimeout* is the number of milliseconds that the calling thread will wait for the muxwait semaphore to clear. If the specified muxwait semaphore is not cleared during this time interval, the request times out.

```
#define INCL_DOSSEMAPHORES    /* Semaphore values          */
#define INCL_WINMESSAGEGR     /*
#include <os2.h>
#include <stdio.h>

#ifndef ERROR_TIMEOUT
#define ERROR_TIMEOUT        640
#define ERROR_INTERRUPT      95
#endif

HMUX  hMux;                /* Muxwait semaphore handle */
ULONG ulTimeout;           /* Number of milliseconds to wait */
```

```

ULONG ulUser;                /* User field for the semaphore that */
                              /* was posted or released (returned) */
ULONG rc;                    /* Return code */

ulTimeout = 60000; /* Wait for a maximum of 1 minute */

rc = WinWaitMuxWaitSem(hmux, ulTimeout, &ulUser);

/* On successful return, the ulUser variable contains the user */
/* identifier of the semaphore that caused the wait to terminate. */
/* If the caller had to wait for all the semaphores within the */
/* muxwait semaphore to clear, then the value corresponds to the */
/* last semaphore within the muxwait semaphore to clear. */
/* If the caller had to wait for any semaphore with the muxwait */
/* semaphore to clear, then the value corresponds to that */
/* semaphore. */

if (rc == ERROR_TIMEOUT)
{
    printf("WinWaitMuxWaitSem call timed out");
    return;
}

if (rc == ERROR_INTERRUPT)
{
    printf("WinWaitMuxWaitSem call was interrupted");
    return;
}

if (rc != 0)
{
    printf("WinWaitMuxWaitSem error: return code = %ld", rc);
    return;
}

```

WinWaitMuxWaitSem - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinWindowFromDC

WinWindowFromDC - Syntax

This function returns the handle of the window corresponding to a particular device context.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
```

```
#include <os2.h>

HDC     hdc; /* Device-context handle. */
HWND    hwnd; /* Window handle. */

hwnd = WinWindowFromDC(hdc);
```

WinWindowFromDC Parameter - hdc

hdc ([HDC](#)) - input
Device-context handle.

The device context must first be opened by the [WinOpenWindowDC](#) function.

WinWindowFromDC Return Value - hwnd

hwnd ([HWND](#)) - returns
Window handle.

NULLHANDLE

Error occurred. For example, the device context has not been opened by the [WinOpenWindowDC](#) function.

Other

Window handle.

WinWindowFromDC - Parameters

hdc ([HDC](#)) - input
Device-context handle.

The device context must first be opened by the [WinOpenWindowDC](#) function.

hwnd ([HWND](#)) - returns
Window handle.

NULLHANDLE

Error occurred. For example, the device context has not been opened by the [WinOpenWindowDC](#) function.

Other

Window handle.

WinWindowFromDC - Errors

Possible returns from [WinGetLastError](#)

PMERR_INV_HPS (0x207F)

An invalid presentation-space handle was specified.

PMERR_INV_HDC (0x207C)

An invalid device-context handle or (micro presentation space) presentation-space handle was specified.

WinWindowFromDC - Related Functions

Related Functions

- [WinEnableWindow](#)
- [WinIsThreadActive](#)
- [WinIsWindow](#)
- [WinIsWindowEnabled](#)
- [WinQueryDesktopWindow](#)
- [WinQueryObjectWindow](#)
- [WinQueryWindowDC](#)
- [WinQueryWindowProcess](#)
- [WinQueryWindowRect](#)
- [WinWindowFromDC](#)
- [WinWindowFromID](#)
- [WinWindowFromPoint](#)

WinWindowFromDC - Example Code

If a device context handle is specified, this example determines which window is associated with that device context.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
HWND hwnd;
HDC hdc;

/* Assume the device context for a window has been opened in      */
/* some other window procedure. We would like to get              */
/* a handle to that window.                                        */

/* This function is called in some other window:                  */
/*   hdc = WinOpenWindowDC(hwnd);                                  */

hwnd = WinWindowFromDC(hdc);
```

WinWindowFromDC - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Errors](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinWindowFromID

WinWindowFromID - Syntax

This function returns the handle of the child window with the specified identity.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndParent; /* Parent-window handle. */
ULONG    id;        /* Identity of the child window. */
HWND    hwnd;        /* Window handle. */

hwnd = WinWindowFromID(hwndParent, id);
```

WinWindowFromID Parameter - hwndParent

hwndParent (**HWND**) - input
Parent-window handle.

WinWindowFromID Parameter - id

id (**ULONG**) - input
Identity of the child window.

It must be greater or equal to 0 and less or equal to 0xFFFF.

WinWindowFromID Return Value - hwnd

hwnd (**HWND**) - returns
Window handle.

NULLHANDLE	No child window of the specified identity exists
Other	Child-window handle.

WinWindowFromID - Parameters

hwndParent ([HWND](#)) - input
Parent-window handle.

id ([ULONG](#)) - input
Identity of the child window.

It must be greater or equal to 0 and less or equal to 0xFFFF.

hwnd ([HWND](#)) - returns
Window handle.

NULLHANDLE
No child window of the specified identity exists

Other
Child-window handle.

WinWindowFromID - Remarks

To obtain the window handle for an item within a dialog box, set *hwndParent* to the dialog-box window's handle, and set *id* to the identity of the item in the dialog template.

WinWindowFromID - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinWindowFromID - Related Functions

Related Functions

- [WinEnableWindow](#)
 - [WinIsThreadActive](#)
 - [WinIsWindow](#)
 - [WinIsWindowEnabled](#)
 - [WinQueryDesktopWindow](#)
 - [WinQueryObjectWindow](#)
 - [WinQueryWindowDC](#)
 - [WinQueryWindowProcess](#)
 - [WinQueryWindowRect](#)
 - [WinWindowFromDC](#)
 - [WinWindowFromID](#)
 - [WinWindowFromPoint](#)
-

WinWindowFromID - Example Code

This example calls WinWindowFromID to get the window handle of the system menu and calls WinSendMsg to send a message to disable the Close menu item.

```
#define INCL_WINFRAMEGR      /* for FID_ definitions. */
#define INCL_WINMENUS        /* for MIA_ definitions. */
#define INCL_WINWINDOWMGR
#include <OS2.H>

HWND hwndSysMenu, hwndDlg;

hwndSysMenu = WinWindowFromID(hwndDlg, FID_SYSMENU);
WinSendMsg(hwndSysMenu, MM_SETITEMATTR,
    MPFROM2SHORT(SC_CLOSE, TRUE),
    MPFROM2SHORT(MIA_DISABLED, MIA_DISABLED));
```

WinWindowFromID - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinWindowFromPoint

WinWindowFromPoint - Syntax

This function finds the window below a specified point, that is a descendant of a specified window.

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndParent;      /* Window handle whose child windows are to be tested. */
PPOINTL   pptlPoint;       /* The point to be tested. */
BOOL      fEnumChildren;   /* Test control. */
HWND      hwndFound;       /* Window handle beneath pptlPoint. */

hwndFound = WinWindowFromPoint(hwndParent,
    pptlPoint, fEnumChildren);
```

WinWindowFromPoint Parameter - hwndParent

hwndParent ([HWND](#)) - input

Window handle whose child windows are to be tested.

HWND_DESKTOP

The desktop-window handle, implying that all main windows are tested. In this instance, *pptlPoint* must be relative to the bottom left corner of the screen.

Other

Parent-window handle.

WinWindowFromPoint Parameter - pptlPoint

pptlPoint ([PPOINTL](#)) - input

The point to be tested.

Specified in window coordinates relative to the window specified by the *hwndParent* parameter.

WinWindowFromPoint Parameter - fEnumChildren

fEnumChildren ([BOOL](#)) - input

Test control.

TRUE

Test all the descendant windows, including child windows of child windows

FALSE

Test only the immediate child windows.

WinWindowFromPoint Return Value - hwndFound

hwndFound ([HWND](#)) - returns

Window handle beneath *pptlPoint*.

NULLHANDLE

pptlPoint is outside *hwndParent*

Parent

pptlPoint is not inside any of the children of *hwndParent*

Other

Window handle is beneath *pptlPoint*.

WinWindowFromPoint - Parameters

hwndParent (HWND)	- input
Window handle whose child windows are to be tested.	
HWND_DESKTOP	The desktop-window handle, implying that all main windows are tested. In this instance, <i>pptlPoint</i> must be relative to the bottom left corner of the screen.
Other	Parent-window handle.
pptlPoint (PPOINTL)	- input
The point to be tested.	
Specified in window coordinates relative to the window specified by the <i>hwndParent</i> parameter.	
fEnumChildren (BOOL)	- input
Test control.	
TRUE	Test all the descendant windows, including child windows of child windows
FALSE	Test only the immediate child windows.
hwndFound (HWND)	- returns
Window handle beneath <i>pptlPoint</i> .	
NULLHANDLE	<i>pptlPoint</i> is outside <i>hwndParent</i>
Parent	<i>pptlPoint</i> is not inside any of the children of <i>hwndParent</i>
Other	Window handle is beneath <i>pptlPoint</i> .

WinWindowFromPoint - Remarks

This function checks only the descendants of the specified window.

WinWindowFromPoint - Errors

Possible returns from [WinGetLastError](#)

PMERR_INVALID_HWND (0x1001)
An invalid window handle was specified.

WinWindowFromPoint - Related Functions

Related Functions

- [WinEnableWindow](#)
- [WinIsThreadActive](#)
- [WinIsWindow](#)
- [WinIsWindowEnabled](#)

- [WinQueryDesktopWindow](#)
- [WinQueryObjectWindow](#)
- [WinQueryWindowDC](#)
- [WinQueryWindowProcess](#)
- [WinQueryWindowRect](#)
- [WinWindowFromDC](#)
- [WinWindowFromID](#)
- [WinWindowFromPoint](#)

WinWindowFromPoint - Example Code

This example calls WinWindowFromPoint to find out if any main windows are beneath point 100,100.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
HWND hwndunderneath;
POINTL point = { 100L, 100L};
hwndunderneath = WinWindowFromPoint(HWND_DESKTOP,
                                     &point,
                                     FALSE); /* do not test the */
                                           /* descendants of */
                                           /* the main */
                                           /* windows. */
```

WinWindowFromPoint - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Errors](#)
- [Remarks](#)
- [Example Code](#)
- [Related Functions](#)
- [Glossary](#)

Message Processing

Messages are processed by window and dialog procedures.

Every window has a window procedure. Windows can also be combined into standard windows or dialog boxes. These are special cases of groups of windows that also have their own procedures. A window or dialog procedure must be capable of processing any message. This can be achieved by delegating some message types to the default window, or dialog, procedures by use of the [WinDefWindowProc](#) and [WinDefDlgProc](#) functions respectively.

Control windows are a special type of child windows. They take the form of objects such as buttons, scroll bars, list boxes, and text entry fields. These child windows process mouse and keyboard input and notify its owner of significant input events. Procedures for these child window controls are inside the Presentation Manager and are often called system-provided window procedures.

All messages have the same form as [QMSG](#). structure.

Message Types

There are two types of window procedure message processing:

- Default window and dialog procedure message processing
- Control window message processing.

These types are described in [Default Window and Dialog Procedure Message Processing](#) and [Control Window Message Processing](#). The messages are described in the message groups found on the Contents.

Default Window and Dialog Procedure Message Processing

These window procedures provide default processing for application window procedures:

- Default window and dialog procedure
- Language support window and dialog procedures, which are used if the application specifies a null window procedure
- Default AVIO window procedure.

These messages are described in [Default Window Procedure Message Processing](#). The system-provided window procedures take no action on messages that are not defined in this chapter, and return NULL.

Control Window Message Processing

Controls are predefined classes of child windows that any application can use for input and output. These control classes are predefined:

WC_BUTTON	Consists of buttons and boxes that the operator can select by clicking the pointing device or using the keyboard. These messages are described in Button Control Window Processing .
WC_CIRCULARSLIDER	Consists of a visual component whose specific purpose is to allow a user to set, display, or modify a value by moving the slider arm around the circular slider dial. Messages are described in Circular Slider Control Window Messages .
WC_COMBOBOX	Consists of an entry field control and a list box control merged into a single control. The list, which is usually limited in size, is displayed below the entry field and offset one dialog box unit to its right. These messages are described in Combination-Box Control Window Processing .
WC_CONTAINER	Consists of a visual component whose specific purpose is to hold objects such as executable programs, word processing files, graphics images, and database records. Messages are described in Container Control Window Processing .
WC_ENTRYFIELD	Consists of a single line of text that the operator can edit. These messages are

described in [Entry Field Control Window Processing](#).

WC_FRAME	Consists of a composite window. These messages are described in Frame Control Window Processing .
WC_LISTBOX	Presents a list of text items from which the operator can make selections. These messages are described in List Box Control Window Processing .
WC_MENU	Presents a list of items, which may be text displayed horizontally as action bars or vertically as pull-down menus. Menus are usually used to provide a command interface to applications. These messages are described in Menu Control Window Processing .
WC_MLE	Consists of a rectangular window that displays multiple lines of text that the operator can edit. When it has the focus, the cursor marks the current <i>insertion</i> or <i>replacement</i> point. These messages are described in Multi-Line Entry Field Control Window Processing .
WC_NOTEBOOK	Consists of a visual component whose specific purpose is to organize information on individual pages so that a user can find and display that information quickly and easily. Messages are described in Notebook Control Window Processing .
WC_SCROLLBAR	Consists of window scroll bars that allow the operator to make a request to scroll the contents of an associated window. These messages are described in Scroll Bar Control Window Processing .
WC_SLIDER	Consists of a visual component whose specific purpose is to allow a user to set, display, or modify a value by moving the slider arm along the slider shaft. Messages are described in Slider Control Window Processing .
WC_SPINBUTTON	Presents a scrollable ring of choices from which the operator can select. These messages are described in Spin Button Control Window Processing .
WC_STATIC	Consists of simple display items that do not respond to keyboard or pointing device events. These messages are described in Static Control Window Processing .
WC_TITLEBAR	Displays the window title or caption and allows the operator to move its owner. These messages are described in Title Bar Control Window Processing .
WC_VALUESET	Consists of a visual component whose specific purpose is to allow a user to select one choice from a group of mutually exclusive choices. A value set can use graphical images (bit maps or icons), as well as colors, text, and numbers, to represent the items that a user can select. Messages are described in Value Set Control Window Processing .

Owner-Notification Messages

Controls are useful because they notify their owners when significant events take place. A control notifies its owner by sending a WM_CONTROL message or by posting a WM_COMMAND or WM_HELP message.

- WM_CONTROL
- WM_COMMAND

Param2 contains information that indicates the source of the WM_COMMAND message:

CMDSRC_PUSHBUTTON	Posted by a pushbutton control
CMDSRC_MENU	Posted by a menu control
CMDSRC_ACCELERATOR	Posted by WinTranslateAccel
CMDSRC_FONTDLG	Posted by a font dialog.
CMDSRC_OTHER	Other source.

- WM_HELP

Param1 contains information that indicates the source of the WM_HELP message:

CMDSRC_PUSHBUTTON	Posted by a pushbutton control
CMDSRC_MENU	Posted by a menu control
CMDSRC_ACCELERATOR	Posted by WinTranslateAccel
CMDSRC_OTHER	Other source.

Notation Conventions

Each message description contains:

Name

The message name; a 2-byte identity unique to a message.

Some message identity values are reserved for the use of the operating system, some are available for use by an application. See **Reserved Messages** under [Default Window Procedure Message Processing](#).

For all messages, the first two or three characters of the name indicate the type of window that is related to the message; for example:

LM	List box control
SBM	Scroll bar control.

Cause

The principal reason that caused the generation of the message.

Parameters

Input and output parameters pertinent to the message.

There are always two parameters (*param1* and *param2*) and one *return* value. Any or all of the parameters can be NULL.

Remarks

An explanation of the relationship between the parameters in the context of the message and an indication of the expected processing of the message.

Default

A definition of how the default window procedures (provided by the system) process the message.

Note: A message is not equivalent to a call of the same name.

Button Control Window Processing

This system-provided window procedure processes the actions on a button control (WC_BUTTON).

For information on button control data see [BTNCDATA](#).

Purpose

A button control is a small rectangular child window representing a button that the operator can "switch" on or off. Button controls can be used alone or in groups, and can either be labeled or appear without text. Button controls typically change appearance when the operator clicks a pointing device on them or pressing the space bar when the button has the keyboard focus.

Buttons can be disabled to prevent them from responding when the operator clicks on them. Disabled buttons are displayed using a different emphasis technique (for example, color or half-toning).

Button Control Styles

These button control styles are available:

BS_AUTOCHECKBOX

An automatic check box automatically toggles its state whenever the user clicks on it.

BS_AUTORADIOBUTTON

When clicked, an automatic radio button automatically checks itself and unchecks all other radio buttons in the same group.

BS_AUTOSIZE

Buttons with this style are sized automatically to make sure the contents fit.

If BS_AUTOSIZE is selected when the button is created, and a -1 is specified for either the parameter of [WinCreateWindow](#), (or when creating the button as a resource) then the button's optimal size is calculated to display the its contents.

BS_AUTO3STATE

An automatic three-state check box automatically toggles its state when the user clicks on it.

BS_BITMAP

Places a bit map instead of text on the push button control. This style works only with the BS_PUSHBUTTON.

BS_CHECKBOX

A check box is a small square with a character string to the right. If it is checked, a small black box appears inside the small square. When the box or string is clicked, by clicking on it with the pointing device or pressing the keyboard spacebar when it is active,

BS_DEFAULT

A BS_DEFAULT pushbutton is one with a thick border box. It has the same properties as a pushbutton. In addition, the user may press a BS_DEFAULT pushbutton by pressing the RETURN or ENTER key. The intention is the same for user-buttons, but the appearance of a BS_DEFAULT userbutton is application defined.

This style can be ORed with the BS_PUSHBUTTON and BS_USERBUTTON styles:

BS_HELP

The button posts a [WM_HELP](#) message rather than a [WM_COMMAND](#) message.

This style can be ORed with the BS_PUSHBUTTON style.

If both BS_HELP and BS_SYSCOMMAND are set, BS_HELP takes precedence.

BS_ICON

Places an icon instead of text on the push button control. This style works only with the BS_PUSHBUTTON style.

BS_MINIICON

This enables miniicons (half the size of normal icons) to be placed on the push button control.

BS_NOBORDER

The pushbutton is displayed without a border drawn around it. There is no other change in the pushbutton's operation.

This style can be ORed with the BS_PUSHBUTTON style.

BS_NOCURSORSELECT

The radio button does not select itself when given the focus as the result of an arrow key or tab key.

This style can be ORed with the BS_AUTORADIOBUTTON style.

BS_NOPOINTERFOCUS

Buttons with this style do not set the focus to themselves when clicked with the pointing device. This enables the cursor to stay on a control for which information is required, rather than moving to the button. This style has no effect on keyboard interaction. The tab key can still be used as usual to move the focus to the button.

This style can be ORed with any of the basic button styles.

BS_NOTEBOOKBUTTON

A notebook button is identical to a pushbutton except that when it is created as a child of a notebook page it becomes a button in the common button area of the notebook page. If the button is not in a notebook page it will be indistinguishable from a pushbutton.

BS_PUSHBUTTON

A pushbutton is a box that contains a string. When a button is pushed, by clicking the pointing device on it or pressing the spacebar when it is active, the parent window is notified.

BS_RADIOBUTTON

A radio button is similar to a check box, but is typically used in groups in which only one button at a time is checked. When a radio button is clicked or a cursor key is pressed to move within the group, it notifies its owner window. It is then up to the owner window to check the clicked radio button and uncheck all the rest, if necessary.

BS_SYSCOMMAND

The button posts a [WM_SYSCOMMAND](#) message rather than a [WM_COMMAND](#) message.

This style can be ORed with the BS_PUSHBUTTON style.

If both BS_HELP and BS_SYSCOMMAND are set, BS_HELP takes

BS_TEXT

This enables both text and a bitmap, icon, or miniicon to be placed on the push button control. This style works only with the BS_PUSHBUTTON style, and should be used in conjunction with BS_BITMAP, BS_ICON or BS_MINIICON.

BS_USERBUTTON

This is an application-definable button. The owner window of this style control receives the additional button style BN_PAINT.

BS_3STATE

A three-state check box is identical to a check box control except that its check box can be half-toned as well as the box being checked or unchecked.

When BS_ICON, BS_MINIICON or BS_BITMAP is selected, the image can be activated by specifying the image ID with the button text string. For instance, to load an icon (#define ICON_ID 300), and display it within a button, the button string is set to "#300."

When BS_ICON, BS_MINIICON or BS_BITMAP is selected along with BS_TEXT, the image can still be activated by specifying the following with a zero-terminated text string. format:

```
"#<image-id>\t<text>"
```

where:

<image-id>	resource id of the icon, miniicon or bitmap
\t	tab character
<text>	zero-terminated button text string

For example, to load an icon (#define ICON_ID 300) and display it with the button text "My Button," the button string is set to "#300\tMy Button." Notice the "\t" is used to separate the text from the image-id. The image is displayed above the text within the button.

Default Colors

The following system colors are used when the system draws button controls:

```
SYSCLR_BUTTONDEFAULT  
SYSCLR_BUTTONLIGHT  
SYSCLR_BUTTONMIDDLE  
SYSCLR_MENUTEXT  
SYSCLR_WINDOW  
SYSCLR_WINDOWFRAME.
```

Some of these defaults can be replaced by using the following presentation parameters in the application resource script file or source code:

```
PP_BACKGROUNDCOLOR  
PP_BORDERCOLOR  
PP_DISABLEDFOREGROUND  
PP_FOREGROUND  
PP_HILITEFOREGROUND.
```

Button Control Notification Messages

These messages are initiated by the button control window to notify its owner of significant events.

WM_COMMAND (in Button Controls)

WM_COMMAND (in Button Controls) - Syntax

For the cause of this message, see [WM_COMMAND](#).

For a description of the parameters, see [WM_COMMAND](#).

Button control sets *uscmd* to the button identity and *ussource* to CMDSRC_PUSHBUTTON.

WM_COMMAND (in Button Controls) - Remarks

The button control generates this message when a push button of style BS_PUSHBUTTON is pressed or when it receives a [BM_CLICK](#) message. The button control posts the message to the queue of the control owner.

WM_COMMAND (in Button Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *uiReserved* to 0.

WM_COMMAND (in Button Controls) - Related Messages

Related Messages

- [WM_COMMAND](#)
-

WM_COMMAND (in Button Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_CONTROL (in Button Controls)

WM_CONTROL (in Button Controls) Field - id

id ([USHORT](#))
Button control identity.

WM_CONTROL (in Button Controls) Field - usnotifycode

usnotifycode ([USHORT](#))
Notification code.

The notification code BN_PAINT is only generated when the button control has a style of BS_USERBUTTON.

The button control uses these notification codes:

BN_CLICKED The button has been pressed.

BN_DBLCLICKED The button has been double-clicked.

BN_PAINT The button requires painting, using one of the following draw states:

BDS_DISABLED The disabled state of the button requires painting.

BDS_HILITED The highlighted state of the button requires painting.

BDS_DEFAULT The default state of the button requires painting.

WM_CONTROL (in Button Controls) Field - flcontrolspect

flcontrolspect ([ULONG](#))
Control-specific information.

When *usnotifycode* is BN_PAINT this parameter is a pointer to a [USERBUTTON](#) structure, otherwise this parameter is the window handle of the button control.

WM_CONTROL (in Button Controls) Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_CONTROL (in Button Controls) - Parameters

id ([USHORT](#))

Button control identity.

usnotifycode ([USHORT](#))

Notification code.

The notification code BN_PAINT is only generated when the button control has a style of BS_USERBUTTON.

The button control uses these notification codes:

BN_CLICKED

The button has been pressed.

BN_DBLCLICKED

The button has been double-clicked.

BN_PAINT

The button requires painting, using one of the following draw states:

BDS_DISABLED

The disabled state of the button requires painting.

BDS_HILITED

The highlighted state of the button requires painting.

BDS_DEFAULT

The default state of the button requires painting.

flcontrolspect ([ULONG](#))

Control-specific information.

When *usnotifycode* is BN_PAINT this parameter is a pointer to a [USERBUTTON](#) structure, otherwise this parameter is the window handle of the button control.

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_CONTROL (in Button Controls) - Syntax

For the cause of this message, see [WM_CONTROL](#).

```
param1
    USHORT id           /* Button control identity. */
    USHORT usnotifycode /* Notification code. */

param2
    ULONG flcontrolspect /* Control-specific information. */
```

WM_CONTROL (in Button Controls) - Remarks

The button control generates this message and sends it to its owner, informing the owner of this event, when:

- Its style is not BS_PUSHBUTTON and the button is pressed.
- It receives a [BM_CLICK](#) message.
- Its style is BS_USERBUTTON and the button is clicked or double clicked.

WM_CONTROL (in Button Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *uiReserved* to 0.

WM_CONTROL (in Button Controls) - Related Messages

Related Messages

- [WM_CONTROL](#)

WM_CONTROL (in Button Controls) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_HELP (in Button Controls)

WM_HELP (in Button Controls) - Syntax

For the cause of this message, see [WM_HELP](#).

For a description of the parameters, see [WM_HELP](#).

Button control sets *uscmd* to the button identity.

WM_HELP (in Button Controls) - Remarks

This message is identical to a [WM_COMMAND](#) message, but implies that the application should respond to this message by displaying help information.

The button control generates this message and posts it to the queue of its owner, if it has the style of BS_HELP and a push button is pressed, or when it receives a [BM_CLICK](#) message.

WM_HELP (in Button Controls) - Default Processing

The default window procedure sends this message to the parent window, if it exists and is not the desktop. Otherwise, it sets *uReserved* to 0.

WM_HELP (in Button Controls) - Related Messages

Related Messages

- [WM_HELP](#)
-

WM_HELP (in Button Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_SYSCOMMAND

WM_SYSCOMMAND - Syntax

For the cause of this message, see [WM_SYSCOMMAND](#).

For a description of the parameters, see [WM_SYSCOMMAND](#).

Button control sets *uscmd* to the button identity.

WM_SYSCOMMAND - Remarks

If the button control is specified with a style of BS_SYSCOMMAND but not with BS_HELP, the button control generates this message and posts it to the queue of its owner when a push button is pressed, or when it receives a [BM_CLICK](#) message.

WM_SYSCOMMAND - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_SYSCOMMAND - Topics

- Select an item:
- [Syntax](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

Button Control Window Messages

This section describes the Button Control Window Procedure actions on receiving the following messages.

BM_AUTOSIZE

BM_AUTOSIZE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

BM_AUTOSIZE Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

BM_AUTOSIZE Return Value - rc

rc (BOOL)

Success indicator.

TRUE

The button was automatically sized according to the size of its contents.

FALSE

The button was not automatically sized.

BM_AUTOSIZE - Parameters

ulReserved (ULONG)

Reserved value, should be 0.

ulReserved (ULONG)

Reserved value, should be 0.

rc (BOOL)

Success indicator.

TRUE

The button was automatically sized according to the size of its contents.

FALSE

The button was not automatically sized.

BM_AUTOSIZE - Syntax

This message causes a button to be resized based on the size of its contents.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

BM_AUTOSIZE - Remarks

This message is used by the notebook control to size buttons placed in the common area. If the button control has a style of BS_NOTEBOOKBUTTON, it responds to this message by setting rc as appropriate. If the button has any other style, the button control

takes no action other than to set *rc* to FALSE.

BM_AUTOSIZE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, except to set *rc* to the default value of FALSE.

BM_AUTOSIZE - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

BM_CLICK

BM_CLICK Field - usUp

- usUp** ([USHORT](#))
Up and down indicator.
- | | |
|-------|---------------------------------------|
| TRUE | Perform the default upclick action |
| FALSE | Perform the default downclick action. |

BM_CLICK Field - ulReserved

- ulReserved** ([ULONG](#))
Reserved value, should be 0.

BM_CLICK Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

BM_CLICK - Parameters

usUp ([USHORT](#))
Up and down indicator.

TRUE	Perform the default upclick action
FALSE	Perform the default downclick action.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

BM_CLICK - Syntax

An application sends this message to cause the effect of the operator clicking a push button.

```
param1
    USHORT    usUp        /* Up and down indicator. */

param2
    ULONG     ulReserved  /* Reserved value, should be 0. */
```

BM_CLICK - Remarks

The button control responds to this message by taking the action that occurs if the button is clicked by the operator. This causes the following messages to be generated:

- A [WM_HELP](#) (in [Button Controls](#)) message, if the button has a style of BS_HELP.
- A [WM_SYSCOMMAND](#) message, if the button has a style of BS_PUSHBUTTON and a style of BS_SYSCOMMAND and not a style of BS_HELP.
- A [WM_COMMAND](#) (in [Button Controls](#)) message, if the button has a style of BS_PUSHBUTTON but not a style of BS_SYSCOMMAND and not a style of BS_HELP.
- A [WM_CONTROL](#) (in [Button Controls](#)) message, whose *usnotifycode* is set to BN_CLICKED, if the button has a style of BS_USERBUTTON, BS_PUSHBUTTON, BS_CHECKBOX, or BS_3STATE, and not a style of BS_SYSCOMMAND or BS_HELP.

BM_CLICK - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *ulReserved* to the default value of 0.

BM_CLICK - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

BM_QUERYCHECK

BM_QUERYCHECK Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

BM_QUERYCHECK Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

BM_QUERYCHECK Return Value - usCheck

usCheck ([USHORT](#))
Check indicator.

0	The button control is in unchecked state.
---	---

- | | |
|---|---|
| 1 | The button control is in checked state. |
| 2 | The button control is in indeterminate state. |

BM_QUERYCHECK - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

usCheck ([USHORT](#))
Check indicator.

- | | |
|---|---|
| 0 | The button control is in unchecked state. |
| 1 | The button control is in checked state. |
| 2 | The button control is in indeterminate state. |

BM_QUERYCHECK - Syntax

This message returns the checked state of a button control.

```
param1
    ULONG    ulReserved /* Reserved value, should be 0. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */
```

BM_QUERYCHECK - Remarks

The button control responds to this message, if it has a style of BS_CHECKBOX, BS_AUTOCHECKBOX, BS_RADIOBUTTON, BS_AUTORADIOBUTTON, BS_3STATE, or BS_AUTO3STATE, by setting *usCheck* as appropriate.

If the button has any other style, the button control takes no action other than to set *usCheck* to 0.

BM_QUERYCHECK - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *usCheck* to the default value of 0.

BM_QUERYCHECK - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

BM_QUERYCHECKINDEX

BM_QUERYCHECKINDEX Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

BM_QUERYCHECKINDEX Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

BM_QUERYCHECKINDEX Return Value - sIndex

sIndex ([SHORT](#))
Radio-button index.

-1	No radio button of the group is checked, or this button control does not have the style BS_RADIOBUTTON or BS_AUTORADIOBUTTON.
Other	Zero-based index of the checked radio button of the group.

BM_QUERYCHECKINDEX - Parameters

ulReserved ([ULONG](#))

Reserved value, should be 0.

ulReserved ([ULONG](#))

Reserved value, should be 0.

sIndex ([SHORT](#))

Radio-button index.

-1

No radio button of the group is checked, or this button control does not have the style BS_RADIOBUTTON or BS_AUTORADIOBUTTON.

Other

Zero-based index of the checked radio button of the group.

BM_QUERYCHECKINDEX - Syntax

This message returns the zero-based index of a checked radio button.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */
param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

BM_QUERYCHECKINDEX - Remarks

The button control responds to this message by setting *sIndex* as appropriate.

This message may be sent to any radio button or autoradio button in a group of buttons. For details of the WS_GROUP style, see [Window Styles](#).

BM_QUERYCHECKINDEX - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sIndex* to the default value of 0.

BM_QUERYCHECKINDEX - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

Reserved value, should be 0.

Reserved value, should be 0.

Highlight indicator.

The button control is displayed in highlighted state.

The button control is displayed in unhighlighted state.

Reserved value, should be 0.

Reserved value, should be 0.

Highlight indicator.

The button control is displayed in highlighted state.

The button control is displayed in unhighlighted state.

BM_QUERYHILITE - Syntax

This message returns the highlighting state of a button control.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

BM_QUERYHILITE - Remarks

The button control responds to this message, if it has a style of BS_PUSHBUTTON, by setting *rc* as appropriate.

If the button has any other style, the button control takes no action other than to set *rc* to FALSE.

BM_QUERYHILITE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, except to set *rc* to the default value of FALSE.

BM_QUERYHILITE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

BM_SETCHECK

BM_SETCHECK Field - uscheck

uscheck (USHORT)
Check state.

- | | |
|---|--|
| 0 | Display the button control in the unchecked state |
| 1 | Display the button control in the checked state |
| 2 | Display a 3-state button control in the indeterminate state. |

BM_SETCHECK Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

BM_SETCHECK Return Value - usoldstate

usoldstate (USHORT)
Old check state of the button control.

- | | |
|---|----------------|
| 0 | Unchecked |
| 1 | Checked |
| 2 | Indeterminate. |

BM_SETCHECK - Parameters

uscheck (USHORT)
Check state.

- | | |
|---|--|
| 0 | Display the button control in the unchecked state |
| 1 | Display the button control in the checked state |
| 2 | Display a 3-state button control in the indeterminate state. |

ulReserved (ULONG)
Reserved value, should be 0.

usoldstate (USHORT)
Old check state of the button control.

- | | |
|---|----------------|
| 0 | Unchecked |
| 1 | Checked |
| 2 | Indeterminate. |

BM_SETCHECK - Syntax

This message sets the checked state of a button control.


```
param1
    USHORT  uscheck      /* Check state. */

param2
    ULONG   ulReserved  /* Reserved value, should be 0. */
```

BM_SETCHECK - Remarks

The button control responds to this message by displaying it in the appropriate state and returning the old state.

If the button control has the style of BS_CHECKBOX, BS_AUTOCHECKBOX, BS_RADIOBUTTON, or BS_AUTORADIOBUTTON, it is displayed in the checked state if *uscheck* is set to 1, or in the unchecked state if it is set to 0 and *usoldstate* is set as appropriate.

If the button control has the style of BS_RADIOBUTTON or BS_AUTORADIOBUTTON, the WS_TABSTOP style is modified. If the resulting state of the button is checked, the WS_TABSTOP style is set, otherwise it is reset.

If the button control has the style of BS_3STATE or BS_AUTO3STATE, it is displayed in the unchecked state if *uscheck* is set to 0, in the checked state if it is set to 1, and in the indeterminate state if it is set to 2 and *usoldstate* is set as appropriate.

If the button control has the style of BS_USERBUTTON, a [WM_CONTROL \(in Button Controls\)](#) message is sent to its owner with *usnotifycode* set to BN_PAINT and *usoldstate* is set as appropriate.

If the button control has any other style, the button control takes no action other than to set *usoldstate* to 0.

BM_SETCHECK - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, except to set *usoldstate* to the default value of 0.

BM_SETCHECK - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

BM_SETDEFAULT

BM_SETDEFAULT Field - usdefault

usdefault (USHORT)
Default state.

TRUE	Display the button control in the default state
FALSE	Display the button control in the nondefault state.

BM_SETDEFAULT Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

BM_SETDEFAULT Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful operation
FALSE	Error occurred.

BM_SETDEFAULT - Parameters

usdefault (USHORT)
Default state.

TRUE	Display the button control in the default state
FALSE	Display the button control in the nondefault state.

ulReserved (ULONG)
Reserved value, should be 0.

rc (BOOL)
Success indicator.

TRUE	Successful operation
FALSE	Error occurred.

BM_SETDEFAULT - Syntax

This message sets the default state of a button control.

```
param1
    USHORT  usdefault /* Default state. */

param2
    ULONG   ulReserved /* Reserved value, should be 0. */
```

BM_SETDEFAULT - Remarks

The button control responds to this message, if it has a style of BS_USERBUTTON or BS_PUSHBUTTON, by displaying the button control in the default or nondefault state as appropriate, and setting *rc* to TRUE.

If the button control has any other style, the button control takes no action other than to set *rc* to FALSE.

BM_SETDEFAULT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

BM_SETDEFAULT - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

BM_SETHILITE

BM_SETHILITE Field - ushilite

ushilite (USHORT)
Highlight indicator.

- TRUE Display the button control in the highlighted state
- FALSE Display the button control in the unhighlighted state.

BM_SETHILITE Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

BM_SETHILITE Return Value - foldstate

foldstate (BOOL)
Old highlight state.

- TRUE The button control was in highlighted state
- FALSE The button control was in unhighlighted state.

BM_SETHILITE - Parameters

ushilite (USHORT)
Highlight indicator.

- TRUE Display the button control in the highlighted state
- FALSE Display the button control in the unhighlighted state.

ulReserved (ULONG)
Reserved value, should be 0.

foldstate (BOOL)
Old highlight state.

- TRUE The button control was in highlighted state
- FALSE The button control was in unhighlighted state.

BM_SETHILITE - Syntax

This message sets the highlight state of a button control.

```
param1
    USHORT    ushilite    /* Highlight indicator. */

param2
    ULONG     ulReserved  /* Reserved value, should be 0. */
```

BM_SETHILITE - Remarks

The button control responds to this message, if it has a style of BS_PUSHBUTTON, BS_CHECKBOX, BS_AUTOCHECKBOX, BS_RADIOBUTTON, BS_AUTORADIOBUTTON, BS_3STATE, or BS_AUTO3STATE, by displaying the button control in the appropriate highlight state and setting *foldstate* as appropriate.

If the style of the Button Control is BS_USERBUTTON, a [WM_CONTROL \(in Button Controls\)](#) message is sent to its owner with *usnotifycode* set to BN_PAINT and with *flcontrolspect* pointing to a [USERBUTTON](#) structure and sets *foldstate* as appropriate.

BM_SETHILITE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *foldstate* to the default value of FALSE.

BM_SETHILITE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_ENABLE (in Button Controls)

WM_ENABLE (in Button Controls) - Syntax

For the cause of this message, see [WM_ENABLE](#).

For a description of the parameters, see [WM_ENABLE](#).

WM_ENABLE (in Button Controls) - Remarks

This message notifies the button control window procedure of a change in the enable state of the button.

WM_ENABLE (in Button Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *u/Reserved* to 0.

WM_ENABLE (in Button Controls) - Related Messages

Related Messages

- [WM_ENABLE](#)
-

WM_ENABLE (in Button Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_MATCHMNEMONIC (in Button Controls)

WM_MATCHMNEMONIC (in Button Controls) - Syntax

For the cause of this message, see [WM_MATCHMNEMONIC](#).

For a description of the parameters, see [WM_MATCHMNEMONIC](#).

WM_MATCHMNEMONIC (in Button Controls) - Remarks

The button control window procedure responds to this message by setting *rc* as appropriate. If MP1 matches the button mnemonic, return *rc* to TRUE.

WM_MATCHMNEMONIC (in Button Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_MATCHMNEMONIC (in Button Controls) - Related Messages

Related Messages

- [WM_MATCHMNEMONIC](#)

WM_MATCHMNEMONIC (in Button Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_QUERYCONVERTPOS (in Button Controls)

WM_QUERYCONVERTPOS (in Button Controls) - Syntax

For the cause of this message, see [WM_QUERYCONVERTPOS](#).

For a description of the parameters, see [WM_QUERYCONVERTPOS](#).

WM_QUERYCONVERTPOS (in Button Controls) - Remarks

The button control window procedure returns QCP_NOCONVERT.

WM_QUERYCONVERTPOS (in Button Controls) - Default Processing

For the default window procedure processing of this message see [WM_QUERYCONVERTPOS](#).

WM_QUERYCONVERTPOS (in Button Controls) - Related Messages

Related Messages

- [WM_QUERYCONVERTPOS](#)
-

WM_QUERYCONVERTPOS (in Button Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_QUERYWINDOWPARAMS (in Button Controls)

WM_QUERYWINDOWPARAMS (in Button Controls) - Syntax

Occurs when an application queries the button control window procedure window parameters.

For a description of the parameters, see [WM_QUERYWINDOWPARAMS](#).

WM_QUERYWINDOWPARAMS (in Button Controls) - Remarks

The button control window procedure responds to this message by passing it to the default window procedure.

WM_QUERYWINDOWPARAMS (in Button Controls) - Default Processing

The default window procedure sets the *cchText*, *cbPresParams*, and *cbCtlData* parameters of the [WNDPARAMS](#) data structure, identified by *pwndparams*, to zero and sets *rc* to FALSE.

WM_QUERYWINDOWPARAMS (in Button Controls) - Related Messages

Related Messages

- [WM_QUERYWINDOWPARAMS](#)
-

WM_QUERYWINDOWPARAMS (in Button Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_SETWINDOWPARAMS (in Button Controls)

WM_SETWINDOWPARAMS (in Button Controls) - Syntax

Occurs when an application sets or changes the button control window procedure window parameters.

For a description of the parameters, see [WM_SETWINDOWPARAMS](#).

WM_SETWINDOWPARAMS (in Button Controls) - Remarks

The button control window procedure responds to this message by passing it to the default window procedure.

WM_SETWINDOWPARAMS (in Button Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_SETWINDOWPARAMS (in Button Controls) - Related Messages

Related Messages

- [WM_SETWINDOWPARAMS](#)
-

WM_SETWINDOWPARAMS (in Button Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

Circular Slider Control Window Messages

The system-provided window procedure processes the actions on a circular control (WC_CIRCULARSLIDER).

Purpose

The circular slider control supports values set in analog rather than digital form. This control is intended to emulate the actual controls of stereo and video components.

The circular slider can be used instead of a linear slider. While, at present, there are no particular guidelines as to when a circular slider should replace a linear slider, the circular slider consumes less space on the screen and, therefore, is practical to represent several controls in the same window. For example, for an audio attributes dialog that has volume, balance, bass, and treble controls, you might want to use a linear slider for the volume control (since it is used frequently); but to conserve space and give a more familiar appearance, the circular slider could be used for the balance, bass, and treble.

Circular Slider Control Styles

These circular slider control styles are available:

CSS_CIRCULARVALUE

Draws a circular thumb, rather than a line, for the value indicator.

CSS_MIDPOINT

Makes the mid-point tick mark larger.

CSS_NOBUTTON

Does not display value buttons.

CSS_NONUMBER

Does not display the value on the dial.

CSS_NOTEXT

Does not display title text under the dial.

CSS_POINTSELECT

Permits the values on the circular slider to change immediately when dragged.

Direct manipulation is performed by using a mouse to click on and drag the circular slider. There are two modes of direct manipulation for the circular slider.

The default direct manipulation mode is to *scroll* to the value indicated by the position of the mouse. This could be important if you used a circular slider for a volume control, for example. Increasing the volume from 0% to 100% too quickly could result in damage to both the user's ears and the equipment.

The other mode of direct manipulation permits the value on the circular slider to change immediately when dragged. This mode is enabled using the CSS_POINTSELECT style bit. When this style is used, the value of the dial can be changed by tracking the value with the mouse, which changes values quickly.

CSS_PROPORTIONALTICKS

Allow the length of the tick marks to be calculated as a percentage of the radius.

CSS_360

Permits the scroll range to extend 360 degrees.

CSS_360 forces the CSS_NONUMBER style on. This is necessary to keep the value indicator from corrupting the number value.

Circular Slider Control Data

See [CSBITMAPDATA](#).

Default Colors

The following system colors are used when the system draws circular slider controls:

SYSCLR_BACKGROUND
SYSCLR_FOREGROUND

Some of these defaults can be replaced by using the following presentation parameters in the application resource script file or source code:

PP_BACKGROUND
PP_BORDER

Circular Slider Control Notification Messages

These messages are initiated by the circular slider control window to notify its owner of significant events.

WM_CONTROL (in Circular Slider Controls)

WM_CONTROL (in Circular Slider Controls) Field - usID

usID ([USHORT](#))

Control-window identity.

The identity of the circular slider that generated the notification.

WM_CONTROL (in Circular Slider Controls) Field - usnotifycode

usnotifycode ([USHORT](#))

Notification code.

The notification codes that indicate what action has occurred.

CSN_SETFOCUS

This code returns a Boolean indicating whether the circular slider control sending the notification message is gaining or losing the focus.

param2 contains TRUE if the control is gaining the focus.

CSN_CHANGED

This code is sent to notify the application that the circular slider value has been changed.

param2 contains the new value of the circular slider.

CSN_TRACKING

This code is sent to notify the application that the circular slider is being tracked by the mouse.

param2 contain the inter-media value of the circular slider.

Inter-media values are not necessarily contiguous.

CSN_QUERYBACKGROUNDCOLOR

This code gives the application the opportunity to set the background color of the circular slider. CLR_* or SYSCLR_* values can be returned for the background color.

param2 is NULL.

WM_CONTROL (in Circular Slider Controls) Field -

ulnotifyspec

ulnotifyspec (ULONG)

Notify control-specific information.

WM_CONTROL (in Circular Slider Controls) Return Value - ulReserved

ulReserved (ULONG)

Reserved value.

WM_CONTROL (in Circular Slider Controls) - Parameters

usID (USHORT)

Control-window identity.

The identity of the circular slider that generated the notification.

usnotifycode (USHORT)

Notification code.

The notification codes that indicate what action has occurred.

CSN_SETFOCUS

This code returns a Boolean indicating whether the circular slider control sending the notification message is gaining or losing the focus.

param2 contains TRUE if the control is gaining the focus.

CSN_CHANGED

This code is sent to notify the application that the circular slider value has been changed.

param2 contains the new value of the circular slider.

CSN_TRACKING

This code is sent to notify the application that the circular slider is being tracked by the mouse.

param2 contain the inter-media value of the circular slider.

Inter-media values are not necessarily contiguous.

CSN_QUERYBACKGROUNDCOLOR

This code gives the application the opportunity to set the background color of the circular slider. CLR_* or SYSCLR_* values can be returned for the background color.

param2 is NULL.

ulnotifyspec (ULONG)

Notify control-specific information.

ulReserved (ULONG)

Reserved value.

WM_CONTROL (in Circular Slider Controls) - Syntax

This message occurs when a control has a significant event to notify to its owner.

```
param1
    USHORT  usID          /* Control-window identity. */
    USHORT  usnotifycode /* Notification code. */

param2
    ULONG   ulnotifyspec /* Notify control-specific information. */
```

WM_CONTROL (in Circular Slider Controls) - Remarks

The circular slider control window procedure generates this message and sends it to its owner, informing the owner of this event.

WM_CONTROL (in Circular Slider Controls) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

WM_CONTROLPOINTER (in Circular Slider Controls)

WM_CONTROLPOINTER (in Circular Slider Controls) - Syntax

For the cause of this message, see [WM_CONTROLPOINTER](#).

For a description of the parameters, see [WM_CONTROLPOINTER](#).

WM_CONTROLPOINTER (in Circular Slider Controls) -

Remarks

For the appropriate remarks, see [WM_CONTROLPOINTER](#).

WM_CONTROLPOINTER (in Circular Slider Controls) - Default Processing

For the default processing, see [WM_CONTROLPOINTER](#).

WM_CONTROLPOINTER (in Circular Slider Controls) - Topics

Select an item:

[Syntax](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

Circular Slider Control Window Messages

This section describes the Circular Slider Control Window Procedure actions on receiving the following messages.

CSM_QUERYINCREMENT

CSM_QUERYINCREMENT Field - ScrollIncr

ScrollIncr ([PUSHORT](#))

The increment value added or subtracted for the value of the control when scrolling.

CSM_QUERYINCREMENT Field - TickIncr

TickIncr ([PUSHORT](#))
The increment value used to draw the tick marks.

CSM_QUERYINCREMENT Return Value - rc

rc ([ULONG](#))
Success indicator.

TRUE	Successful completion
FALSE	Errors occurred.

CSM_QUERYINCREMENT - Parameters

ScrollIncr ([PUSHORT](#))
The increment value added or subtracted for the value of the control when scrolling.

TickIncr ([PUSHORT](#))
The increment value used to draw the tick marks.

rc ([ULONG](#))
Success indicator.

TRUE	Successful completion
FALSE	Errors occurred.

CSM_QUERYINCREMENT - Syntax

This message queries the increments used to scroll the value and draw the tick marks.

```
param1
    PUSHORT ScrollIncr /* The increment value added or subtracted for the value of the control when scroll
param2
    PUSHORT TickIncr   /* The increment value used to draw the tick marks. */
```

CSM_QUERYINCREMENT - Topics

Select an item:

CSM_QUERYRADIUS

CSM_QUERYRADIUS Field - uRadius

uRadius ([PUSHORT](#))

The radius of the circular slider.

CSM_QUERYRADIUS Field - ulReserved

ulReserved ([ULONG](#))

Reserved value.

CSM_QUERYRADIUS Return Value - rc

rc ([ULONG](#))

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

CSM_QUERYRADIUS - Parameters

uRadius ([PUSHORT](#))

The radius of the circular slider.

ulReserved ([ULONG](#))

Reserved value.

rc ([ULONG](#))

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

CSM_QUERYRADIUS - Syntax

This message queries the current radius of the circular slider.

```
param1
    PUSHORT  uRadius      /* The radius of the circular slider. */

param2
    ULONG     ulReserved  /* Reserved value. */
```

CSM_QUERYRADIUS - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Glossary](#)

CSM_QUERYRANGE

CSM_QUERYRANGE Field - pLow

pLow ([PSHORT](#))
The low range value.

CSM_QUERYRANGE Field - pHigh

pHigh ([PSHORT](#))

The high range value.

CSM_QUERYRANGE Return Value - rc

rc ([ULONG](#))

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

CSM_QUERYRANGE - Parameters

pLow ([PSHORT](#))

The low range value.

pHigh ([PSHORT](#))

The high range value.

rc ([ULONG](#))

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

CSM_QUERYRANGE - Syntax

This message queries the value range of the control.

```
param1
    PSHORT pLow    /* The low range value. */

param2
    PSHORT pHigh    /* The high range value. */
```

CSM_QUERYRANGE - Topics

Select an item:

[Syntax](#)

CSM_QUERYVALUE

CSM_QUERYVALUE Field - pValue

pValue ([PSHORT](#))

The value of the control.

CSM_QUERYVALUE Field - ulReserved

ulReserved ([ULONG](#))

Reserved value.

CSM_QUERYVALUE Return Value - rc

rc ([ULONG](#))

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

CSM_QUERYVALUE - Parameters

pValue ([PSHORT](#))

The value of the control.

ulReserved ([ULONG](#))

Reserved value.

rc ([ULONG](#))

Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

CSM_QUERYVALUE - Syntax

This message queries the value of the control.

```
param1
    PSHORT  pValue      /* The value of the control. */

param2
    ULONG   ulReserved /* Reserved value. */
```

CSM_QUERYVALUE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

CSM_SETBITMAPDATA

CSM_SETBITMAPDATA Field - pCSBitmapData

pCSBitmapData ([PCSBITMAPDATA](#))
 The structure defining button bit maps.

CSM_SETBITMAPDATA Field - ulReserved

ulReserved ([ULONG](#))
 Reserved value.

CSM_SETBITMAPDATA Return Value - rc

rc (ULONG)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

CSM_SETBITMAPDATA - Parameters

pCSBitmapData (PCSBITMAPDATA)
The structure defining button bit maps.

ulReserved (ULONG)
Reserved value.

rc (ULONG)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

CSM_SETBITMAPDATA - Syntax

This message is used to change the bit maps for the plus and minus buttons. For example, you might want to use left or right arrows. The optimal size for these bit maps is 10 x 10 pels.

```
param1
    PCSBITMAPDATA  pCSBitmapData  /* The structure defining button bit maps. */

param2
    ULONG          ulReserved      /* Reserved value. */
```

CSM_SETBITMAPDATA - Remarks

The optimal size for these bit maps is 10 x 10 pels. Other bit maps are stretched to the necessary size.

CSM_SETBITMAPDATA - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Glossary](#)

CSM_SETINCREMENT

CSM_SETINCREMENT Field - usScrollIncr

usScrollIncr ([USHORT](#))
Scroll increment.

This is the number by which the current value is incremented or decremented when one of the circular slider control button is selected.

CSM_SETINCREMENT Field - usTickIncr

usTickIncr ([USHORT](#))
Tick mark increment.

This represents the number of tick marks to "skip" before drawing tick marks around the circular slider.

CSM_SETINCREMENT Return Value - rc

rc ([ULONG](#))
Success indicator.

- | | |
|-------|-----------------------|
| TRUE | Successful completion |
| FALSE | Error occurred. |

CSM_SETINCREMENT - Parameters

usScrollIncr ([USHORT](#))

Scroll increment.

This is the number by which the current value is incremented or decremented when one of the circular slider control button is selected.

usTickIncr ([USHORT](#))

Tick mark increment.

This represents the number of tick marks to "skip" before drawing tick marks around the circular slider.

rc ([ULONG](#))

Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

CSM_SETINCREMENT - Syntax

This message sets the scroll and tick mark increments of the control.

```
param1
    USHORT  usScrollIncr  /* Scroll increment. */
param2
    USHORT  usTickIncr    /* Tick mark increment. */
```

CSM_SETINCREMENT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Glossary](#)

CSM_SETRANGE

CSM_SETRANGE Field - Low

Low (SHORT)
The minimum value of the circular slider.

CSM_SETRANGE Field - High

High (SHORT)
The maximum value of the circular slider.

CSM_SETRANGE Return Value - rc

rc (ULONG)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

CSM_SETRANGE - Parameters

Low (SHORT)
The minimum value of the circular slider.

High (SHORT)
The maximum value of the circular slider.

rc (ULONG)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

CSM_SETRANGE - Syntax

This message sets the range of values which the control sends to the application via CSN_TRACKING and CSN_CHANGE messages.

```
param1      SHORT  Low      /* The minimum value of the circular slider. */  
param2      SHORT  High     /* The maximum value of the circular slider. */
```

CSM_SETRANGE - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Glossary](#)

CSM_SETVALUE

CSM_SETVALUE Field - Value

Value (**SHORT**)

The new value to which to set the circular slider.

CSM_SETVALUE Field - ulReserved

ulReserved (**ULONG**)

Reserved value.

CSM_SETVALUE Return Value - rc

rc (**ULONG**)

Success indicator.

TRUE

Successful completion.

FALSE

Error occurred.

CSM_SETVALUE - Parameters

Value ([SHORT](#))

The new value to which to set the circular slider.

ulReserved ([ULONG](#))

Reserved value.

rc ([ULONG](#))

Success indicator.

TRUE

Successful completion.

FALSE

Error occurred.

CSM_SETVALUE - Syntax

This message sets the current value of the circular slider control.

```
param1
    SHORT Value          /* The new value to which to set the circular slider. */
param2
    ULONG ulReserved     /* Reserved value. */
```

CSM_SETVALUE - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Glossary](#)

WM_CHAR (in Circular Slider Controls)

WM_CHAR (in Circular Slider Controls) - Syntax

For the cause of this message, see [WM_CHAR](#).
For a description of the parameters, see [WM_CHAR](#).

WM_CHAR (in Circular Slider Controls) - Remarks

The slider control window procedure responds to this message by sending it to its owner if it has not processed the key stroke. This is the most common means by which the input focus is switched around the various controls in a dialog box.

The keystrokes processed by a circular slider control are:

Left Arrow	Moves the slider arm left one increment.
Right Arrow	Moves the slider arm right one increment.

A circular slider control only processes left and right arrow keystrokes. These keys move the slider arm one increment to the left or right.

WM_CHAR (in Circular Slider Controls) - Default Processing

For a description of the default processing, see [WM_CHAR](#).

WM_CHAR (in Circular Slider Controls) - Topics

Select an item:
[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_PRESPARAMCHANGED (in Circular Slider Controls)

WM_PRESPARAMCHANGED (in Circular Slider Controls) Field - attrtype

attrtype ([ULONG](#))
Attribute type.

Presentation parameter attribute identity. The following presentation parameters are initialized by the slider control. The initial value of each is shown in the following list:

PP_FOREGROUND_COLOR or PP_FOREGROUND_COLOR_INDEX
Item foreground color; used when displaying text and bit maps. This color is initialized to
SYSCLR_WINDOWTEXT.
PP_BACKGROUND_COLOR or PP_BACKGROUND_COLOR_INDEX
Slider background color; used for entire control as the background. This color is initialized to
SYSCLR_WINDOW.

WM_PRESPARAMCHANGED (in Circular Slider Controls) Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_PRESPARAMCHANGED (in Circular Slider Controls) Return Value - ulReserved

ulReserved (ULONG)
Reserved value, must be 0.

WM_PRESPARAMCHANGED (in Circular Slider Controls) - Parameters

attrtype (ULONG)
Attribute type.

Presentation parameter attribute identity. The following presentation parameters are initialized by the slider control. The initial value of each is shown in the following list:

PP_FOREGROUND_COLOR or PP_FOREGROUND_COLOR_INDEX
Item foreground color; used when displaying text and bit maps. This color is initialized to
SYSCLR_WINDOWTEXT.
PP_BACKGROUND_COLOR or PP_BACKGROUND_COLOR_INDEX
Slider background color; used for entire control as the background. This color is initialized to
SYSCLR_WINDOW.

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, must be 0.

WM_PRESPARAMCHANGED (in Circular Slider Controls) -

Syntax

For the cause of this message, see [WM_PRESPARAMCHANGED](#).

```
param1
    ULONG attrtype    /* Attribute type. */

param2
    ULONG ulReserved  /* Reserved value, should be 0. */
```

WM_PRESPARAMCHANGED (in Circular Slider Controls) - Remarks

The application uses this message to notify the slider that a given inherited presentation parameter has changed.

WM_PRESPARAMCHANGED (in Circular Slider Controls) - Default Processing

For a description of the default processing, see [WM_PRESPARAMCHANGED](#).

WM_PRESPARAMCHANGED (in Circular Slider Controls) - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

WM_QUERYWINDOWPARAMS (in Circular Slider Controls)

WM_QUERYWINDOWPARAMS (in Circular Slider Controls)

Field - pwndparams

pwndparams (PWNDPARAMS)

Pointer to a [WNDPARAMS](#) window parameter structure.

This structure contains:

status ([ULONG](#))

Window parameter selection.

Identifies the window parameters that are to be queried. Valid values for the slider control are:

WPM_TEXT

Window text.

WPM_CCHTEXT

Window text length.

The flags in the **status** field are unchanged by this processing.

length ([ULONG](#))

Length of the window text.

text ([PSZ](#))

Window text.

presparamslength ([ULONG](#))

Length of presentation parameters.

presparams ([PVOID](#))

Presentation parameters.

ctldatalength ([ULONG](#))

Length of window class-specific data.

ctldata ([PVOID](#))

Window class-specific data.

WM_QUERYWINDOWPARAMS (in Circular Slider Controls) Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_QUERYWINDOWPARAMS (in Circular Slider Controls) Return Value - rc

rc ([BOOL](#))

Success indicator.

TRUE

Successful completion.

FALSE

Error occurred.

WM_QUERYWINDOWPARAMS (in Circular Slider Controls) -

Parameters

pwndparams ([PWNDPARAMS](#))
Pointer to a [WNDPARAMS](#) window parameter structure.

This structure contains:

status ([ULONG](#))
Window parameter selection.

Identifies the window parameters that are to be queried. Valid values for the slider control are:

[WPM_TEXT](#)
Window text.

[WPM_CCHTEXT](#)
Window text length.

The flags in the **status** field are unchanged by this processing.

length ([ULONG](#))
Length of the window text.

text ([PSZ](#))
Window text.

presparamslength ([ULONG](#))
Length of presentation parameters.

presparams ([PVOID](#))
Presentation parameters.

ctldatalength ([ULONG](#))
Length of window class-specific data.

ctldata ([PVOID](#))
Window class-specific data.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Success indicator.

TRUE
Successful completion.

FALSE
Error occurred.

WM_QUERYWINDOWPARAMS (in Circular Slider Controls) - Syntax

For the cause of this message, see [WM_QUERYWINDOWPARAMS](#).

```
param1
    PWNDPARAMS pwndparams /* Pointer to a WNDPARAMS window parameter structure. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

WM_QUERYWINDOWPARAMS (in Circular Slider Controls) -

Remarks

The slider control window procedure responds to this message by returning the information in the buffer provided. If this message is sent to a slider window of another process, the information in, or identified by, the value of the *pwndparams* field must be in memory shared by both processes.

WM_QUERYWINDOWPARAMS (in Circular Slider Controls) - Default Processing

For a description of the default processing, see [WM_QUERYWINDOWPARAMS](#).

WM_QUERYWINDOWPARAMS (in Circular Slider Controls) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_SETWINDOWPARAMS (in Circular Slider Controls)

WM_SETWINDOWPARAMS (in Circular Slider Controls) Field - pwndparams

pwndparams ([PWNDPARAMS](#))

Pointer to a [WNDPARAMS](#) window parameter structure.

This structure contains:

status ([ULONG](#))

Window parameter selection.

Identifies the window parameters that are to be set. The valid value for the slider control is:

WPM_TEXT

Window text.

The flags in the **status** field are cleared as each item is processed. If the call is successful, the **status** field is 0. If any item has not been processed, the flag for that item remains set.

length ([ULONG](#))

text (PSZ)	Length of the window text.
presparamslength (ULONG)	Window text.
presparams (PVOID)	Length of presentation parameters.
ctldatalength (ULONG)	Presentation parameters.
ctldata (PVOID)	Length of window class-specific data.
	Window class-specific data.

WM_SETWINDOWPARAMS (in Circular Slider Controls) Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_SETWINDOWPARAMS (in Circular Slider Controls) Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful operation
FALSE	Error occurred.

WM_SETWINDOWPARAMS (in Circular Slider Controls) - Parameters

pwndparams (PWNDPARAMS)
Pointer to a **WNDPARAMS** window parameter structure.

This structure contains:

status (ULONG)	Window parameter selection.
	Identifies the window parameters that are to be set. The valid value for the slider control is: WPM_TEXT
	Window text.
length (ULONG)	The flags in the status field are cleared as each item is processed. If the call is successful, the status field is 0. If any item has not been processed, the flag for that item remains set.

text (PSZ)	Length of the window text.
presparamslength (ULONG)	Window text.
presparams (PVOID)	Length of presentation parameters.
ctldatalength (ULONG)	Presentation parameters.
ctldata (PVOID)	Length of window class-specific data.
	Window class-specific data.

ulReserved (ULONG)
Reserved value, should be 0.

rc (BOOL)
Success indicator.

TRUE	Successful operation
FALSE	Error occurred.

WM_SETWINDOWPARAMS (in Circular Slider Controls) - Syntax

For the cause of this message, see [WM_SETWINDOWPARAMS](#).

```
param1
    PWNDPARAMS pwndparams /* Pointer to a WNDPARAMS window parameter structure. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

WM_SETWINDOWPARAMS (in Circular Slider Controls) - Remarks

If this message is sent to a slider window of another process, the information in, or identified by, the value of the *pwndparams* field must be in memory shared by both processes.

WM_SETWINDOWPARAMS (in Circular Slider Controls) - Default Processing

For a description of the default processing, see [WM_SETWINDOWPARAMS](#).

WM_SETWINDOWPARAMS (in Circular Slider Controls) -

Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

Clipboard Message Processing

The clipboard is used by the end-user to transfer data between Presentation Manager* (PM*) applications using the following operations:

Cut	Remove from a window, leaving a gap in the source, and save for later use.
Copy	Copy from a window, leaving the source intact, and save for later use.
Paste	Paste the cut or copied data into the window of an application (the target).

Clipboard Messages

This section contains the messages used by the Clipboard.

WM_DESTROYCLIPBOARD

WM_DESTROYCLIPBOARD Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_DESTROYCLIPBOARD Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_DESTROYCLIPBOARD Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_DESTROYCLIPBOARD - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_DESTROYCLIPBOARD - Syntax

This message is sent to the clipboard owner when the clipboard is emptied through a call to [WinEmptyClipbrd](#).

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

WM_DESTROYCLIPBOARD - Remarks

If there is any data that has been set with the CFI_OWNERFREE flag, the clipboard owner must release the data at this time.

WM_DESTROYCLIPBOARD - Default Processing

None.

WM_DESTROYCLIPBOARD - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_DRAWCLIPBOARD

WM_DRAWCLIPBOARD Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_DRAWCLIPBOARD Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_DRAWCLIPBOARD Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_DRAWCLIPBOARD - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_DRAWCLIPBOARD - Syntax

This message is sent to the clipboard viewer window whenever the contents of the clipboard change; that is, as a result of the [WinCloseClipbrd](#) function following a call to [WinSetClipbrdData](#).

```
param1
    ULONG   ulReserved /* Reserved value, should be 0. */

param2
    ULONG   ulReserved /* Reserved value, should be 0. */
```

WM_DRAWCLIPBOARD - Default Processing

None.

WM_DRAWCLIPBOARD - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Glossary](#)

WM_HSCROLLCLIPBOARD

WM_HSCROLLCLIPBOARD Field - hwndViewer

hwndViewer ([HWND](#))
Handle.

This contains a handle to the clipboard application window.

WM_HSCROLLCLIPBOARD Field - sposScroll

sposScroll (SHORT)

Scroll position.

The position is either:

- 0 *scodeScroll* is other than SB_SLIDERPOSITION
- Other The position of the slider when *scodeScroll* is SB_SLIDERPOSITION.

WM_HSCROLLCLIPBOARD Field - scodeScroll

scodeScroll (SHORT)

Scroll-bar code.

This is one of the SB_* scroll-bar codes as defined in [WM_HSCROLL \(in Horizontal Scroll Bars\)](#).

- SB_LINELEFT Sent if the operator clicks the left arrow of the scroll bar, or presses the VK_LEFT key.
- SB_LINERIGHT Sent if the operator clicks the right arrow of the scroll bar, or presses the VK_RIGHT key.
- SB_PAGELEFT Sent if the operator clicks the area to the left of the slider, or presses the VK_PAGELEFT key.
- SB_PAGERIGHT Sent if the operator clicks the area to the right of the slider, or presses the VK_PAGERIGHT key.
- SB_SLIDERPOSITION Sent to indicate the final position of the slider. *sposScroll* contains the final position of the slider.
- SB_SLIDERTRACK Sent every time the slider position changes if the operator moves the scroll bar slider with the pointer device.
- SB_ENDSCROLL Sent when the operator has finished scrolling, but only if the operator has not been doing any absolute slider positioning.

WM_HSCROLLCLIPBOARD Return Value - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

WM_HSCROLLCLIPBOARD - Parameters

hwndViewer ([HWND](#))
Handle.

This contains a handle to the clipboard application window.

sposScroll ([SHORT](#))
Scroll position.

The position is either:

0

scodeScroll is other than SB_SLIDERPOSITION

Other

The position of the slider when *scodeScroll* is SB_SLIDERPOSITION.

scodeScroll ([SHORT](#))
Scroll-bar code.

This is one of the SB_* scroll-bar codes as defined in [WM_HSCROLL](#) (in [Horizontal Scroll Bars](#)).

SB_LINELEFT

Sent if the operator clicks the left arrow of the scroll bar, or presses the VK_LEFT key.

SB_LINERIGHT

Sent if the operator clicks the right arrow of the scroll bar, or presses the VK_RIGHT key.

SB_PAGELEFT

Sent if the operator clicks the area to the left of the slider, or presses the VK_PAGELEFT key.

SB_PAGERIGHT

Sent if the operator clicks the area to the right of the slider, or presses the VK_PAGERIGHT key.

SB_SLIDERPOSITION

Sent to indicate the final position of the slider. *sposScroll* contains the final position of the slider.

SB_SLIDERTRACK

Sent every time the slider position changes if the operator moves the scroll bar slider with the pointer device.

SB_ENDSCROLL

Sent when the operator has finished scrolling, but only if the operator has not been doing any absolute slider positioning.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_HSCROLLCLIPBOARD - Syntax

This message is sent to the clipboard-owner window when the clipboard contains a data handle for the CFI_OWNERDISPLAY format, and there is an event in the clipboard viewer's horizontal scroll bar.

```
param1
    HWND    hwndViewer    /* Handle. */

param2
    SHORT    sposScroll    /* Scroll position. */
    SHORT    scodeScroll    /* Scroll-bar code. */
```

WM_HSCROLLCLIPBOARD - Remarks

The clipboard owner is responsible for displaying the clipboard contents. The clipboard owner should use [WinInvalidateRect](#) or repaint as desired. The scroll-bar position is also reset.

WM_HSCROLLCLIPBOARD - Default Processing

None.

WM_HSCROLLCLIPBOARD - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_PAINTCLIPBOARD

WM_PAINTCLIPBOARD Field - hwndViewer

hwndViewer ([HWND](#))
Handle.

This is a handle to the clipboard application window.

WM_PAINTCLIPBOARD Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_PAINTCLIPBOARD Return Value - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_PAINTCLIPBOARD - Parameters

hwndViewer ([HWND](#))

Handle.

This is a handle to the clipboard application window.

ulReserved ([ULONG](#))

Reserved value, should be 0.

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_PAINTCLIPBOARD - Syntax

This message is sent when the clipboard contains a data handle with the CFI_OWNERDISPLAY information flag set.

```
param1
    HWND    hwndViewer    /* Handle. */

param2
    ULONG    ulReserved    /* Reserved value, should be 0. */
```

WM_PAINTCLIPBOARD - Remarks

As the clipboard owner is responsible for displaying the clipboard contents, this message notifies the clipboard application that its client area needs repainting. The WM_PAINTCLIPBOARD message is sent to the owner of the clipboard to request repainting of all or part of the client area of the clipboard application.

Note: To determine whether the entire client area needs repainting or just a portion of it, the clipboard owner must compare the dimensions of the drawing area to the dimensions given in the most recent [WM_SIZECLIPBOARD](#) message.

WM_PAINTCLIPBOARD - Default Processing

None.

WM_PAINTCLIPBOARD - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_RENDERALLFMTS

WM_RENDERALLFMTS Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_RENDERALLFMTS Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_RENDERALLFMTS Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_RENDERALLFMTS - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_RENDERALLFMTS - Syntax

This message is sent to the application that owns the clipboard while the application is being destroyed.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

WM_RENDERALLFMTS - Remarks

The application renders the clipboard data in all formats it is capable of generating and passes a handle to each format to [WinSetClipbrdData](#). This ensures that the data in the clipboard can be rendered even though the application has been destroyed.

WM_RENDERALLFMTS - Default Processing

None.

WM_RENDERALLFMTS - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_RENDERFMT

WM_RENDERFMT Field - usfmt

usfmt (USHORT)

Data format.

This is the format of the data to be rendered.

CF_BITMAP

A bit map.

CF_DSPBITMAP

A bit-map representation of a private data format.

CF_DSPMETAFILE

A metafile representation of a private data format.

CF_DSPTEXT

A textual representation of a private data format.

CF_METAFILE

A metafile.

CF_TEXT

An array of text characters.

WM_RENDERFMT Field - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

WM_RENDERFMT Return Value - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

WM_RENDERFMT - Parameters

usfmt (USHORT)

Data format.

This is the format of the data to be rendered.

CF_BITMAP

A bit map.

CF_DSPBITMAP

A bit-map representation of a private data format.

CF_DSPMETAFILE

A metafile representation of a private data format.

CF_DSPTTEXT

A textual representation of a private data format.

CF_METAFILE

A metafile.

CF_TEXT

An array of text characters.

ulReserved (ULONG)

Reserved value, should be 0.

ulReserved (ULONG)

Reserved value, should be 0.

WM_RENDERFMT - Syntax

This message is a request to the clipboard owner to render the data of the format specified in *usfmt*.

```
param1
    USHORT    usfmt        /* Data format. */

param2
    ULONG     ulReserved   /* Reserved value, should be 0. */
```

WM_RENDERFMT - Remarks

The data is rendered into a global handle, which is then set into the clipboard with [WinSetClipbrdData](#).

WM_RENDERFMT - Default Processing

None.

WM_RENDERFMT - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

WM_SIZECLIPBOARD

WM_SIZECLIPBOARD Field - hwndViewer

hwndViewer ([HWND](#))
Handle of viewer window.

WM_SIZECLIPBOARD Field - ppaint

ppaint ([PRECTL](#))
Rectangle to be re-painted.

WM_SIZECLIPBOARD Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_SIZECLIPBOARD - Parameters

hwndViewer ([HWND](#))
Handle of viewer window.

ppaint ([PRECTL](#))
Rectangle to be re-painted.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_SIZECLIPBOARD - Syntax

This message is sent when the clipboard contains a data handle for the CFI_OWNERDISPLAY format, and the clipboard application window has changed size.

```
param1
    HWND    hwndViewer /* Handle of viewer window. */

param2
    PRECTL  ppaint      /* Rectangle to be re-painted. */
```

WM_SIZECLIPBOARD - Default Processing

The default window procedure takes no action on this message except to set *ulReserved* to 0.

WM_SIZECLIPBOARD - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_VSCROLLCLIPBOARD

WM_VSCROLLCLIPBOARD Field - hwndViewer

hwndViewer ([HWND](#))
Handle.

This contains a handle to the clipboard application window.

WM_VSCROLLCLIPBOARD Field - sposScroll

sposScroll ([SHORT](#))

Scroll position.

The position is either:

0

scodeScroll is other than SB_SLIDERPOSITION

Other

The position of the slider when *scodeScroll* is SB_SLIDERPOSITION.

WM_VSCROLLCLIPBOARD Field - *scodeScroll*

scodeScroll ([SHORT](#))

Scroll-bar code.

This is one of the SB_* scroll-bar codes as defined in [WM_VSCROLL](#).

SB_LINEUP

Sent if the operator clicks on the up arrow of the scroll bar, or presses the VK_UP key.

SB_LINEDOWN

Sent if the operator clicks on the down arrow of the scroll bar, or presses the VK_DOWN key.

SB_PAGEUP

Sent if the operator clicks on the area above the slider, or presses the VK_PAGEUP key.

SB_PAGEDOWN

Sent if the operator clicks on the area below the slider, or presses the VK_PAGEDOWN key.

SB_SLIDERPOSITION

Sent to indicate the final position of the slider.

SB_SLIDERTRACK

If the operator moves the scroll bar slider with the pointer device, this is sent every time the slider position changes.

SB_ENDSCROLL

Sent when the operator has finished scrolling, but only if the operator has not been doing any absolute slider positioning.

WM_VSCROLLCLIPBOARD Return Value - *ulReserved*

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_VSCROLLCLIPBOARD - Parameters

hwndViewer ([HWND](#))

Handle.

This contains a handle to the clipboard application window.

sposScroll ([SHORT](#))

Scroll position.

The position is either:

0

scodeScroll is other than SB_SLIDERPOSITION

Other

The position of the slider when *scodeScroll* is SB_SLIDERPOSITION.

scodeScroll ([SHORT](#))

Scroll-bar code.

This is one of the SB_* scroll-bar codes as defined in [WM_VSCROLL](#).

SB_LINEUP

Sent if the operator clicks on the up arrow of the scroll bar, or presses the VK_UP key.

SB_LINEDOWN

Sent if the operator clicks on the down arrow of the scroll bar, or presses the VK_DOWN key.

SB_PAGEUP

Sent if the operator clicks on the area above the slider, or presses the VK_PAGEUP key.

SB_PAGEDOWN

Sent if the operator clicks on the area below the slider, or presses the VK_PAGEDOWN key.

SB_SLIDERPOSITION

Sent to indicate the final position of the slider.

SB_SLIDERTRACK

If the operator moves the scroll bar slider with the pointer device, this is sent every time the slider position changes.

SB_ENDSCROLL

Sent when the operator has finished scrolling, but only if the operator has not been doing any absolute slider positioning.

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_VSCROLLCLIPBOARD - Syntax

This message is sent to the clipboard owner window when the clipboard contains a data handle for the CFI_OWNERDISPLAY format, and there is an event in the clipboard viewer's vertical scroll bar.

```
param1
    HWND    hwndViewer    /* Handle. */

param2
    SHORT    sposScroll    /* Scroll position. */
    SHORT    scodeScroll    /* Scroll-bar code. */
```

WM_VSCROLLCLIPBOARD - Remarks

The clipboard owner is responsible for displaying the clipboard contents. The clipboard owner should use [WinInvalidateRect](#) or repaint as desired. The scroll bar position is also reset.

WM_VSCROLLCLIPBOARD - Default Processing

None.

WM_VSCROLLCLIPBOARD - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

Combination-Box Control Window Processing

This system-provided window procedure processes the actions on a prompted entry field (combination-box) control (WC_COMBOBOX).

Purpose

A combination-box consists of an entry field control and a list box control merged into a single control. The list, which is usually limited in size, is displayed below the entry field, and offset one dialog-box unit to its right.

When the combination-box control has the focus, the text in the entry field is given selected emphasis and, if the list box control has a matching entry, it is scrolled to show that match at the top of the list.

A combination-box, while sometimes only showing the entryfield, also owns the area occupied by the invisible list box. Another window can and will be clipped to it if they have clipping flags set.

Combination Box Control Styles

These combination-box control styles are available:

CBS_SIMPLE

Both the entry field control and the list box control are visible. When the selection changes in the list box control, the text of the selected item in the list box control is placed in the entry field. Also, the text in the entry field is completed by extending the text of the entry field with the closest match from the list box.

CBS_DROPDOWN

Inherits all the properties of a combination-box control with a style of CBS_SIMPLE and, in addition, the list box control is hidden until the user requests that it should be displayed.

CBS_DROPDOWNLIST

In which the entry field control is replaced by a static control, that displays the current selection from the list box control. The user must explicitly cause the display of the list box control in order to make alternative selections in the list box.

Default Colors

The following system colors are used when the system draws combination-box controls:

```
SYSCLR_WINDOWFRAME
SYSCLR_ENTRYFIELD
SYSCLR_WINDOW
SYSCLR_BUTTONMIDDLE
SYSCLR_BUTTONDARK
SYSCLR_BUTTONLIGHT
SYSCLR_OUTPUTTEXT
SYSCLR_WINDOWTEXT
SYSCLR_HIGHLIGHTFOREGROUND
SYSCLR_HIGHLIGHTBACKGROUND
SYSCLR_FIELDBACKGROUND
SYSCLR_WINDOWFRAME.
```

Some of these defaults can be replaced by using the following presentation parameters in the application resource script file or source code:

```
PP_FOREGROUNDCOLOR
PP_DISABLEDFOREGROUNDCOLOR
PP_HIGHLIGHTFOREGROUNDCOLOR
PP_FONTNAMESIZE
PP_BORDERCOLOR.
```

Combo Box Control Notification Messages

The combo box control uses most of the same window messages as the entry field control and the list box control to notify its owner of significant events.

WM_CONTROL (in Combination Boxes)

WM_CONTROL (in Combination Boxes) Field - usid

usid (**USHORT**)
Control window identity.

WM_CONTROL (in Combination Boxes) Field - usnotifycode

usnotifycode (**USHORT**)
Notify code.

CBN_EFCHANGE
The content of the entry field control has changed, and the change has been displayed on the screen.

CBN_MEMERROR

The entry field control cannot allocate the storage necessary to accommodate window text of the length implied by the [EM_SETTEXTLIMIT](#) message.

CBN_EFSCROLL

The entry field control is about to scroll horizontally. This can happen in these circumstances:

- The application has issued a [WinScrollWindow](#) call.
- The content of the entry field control has changed.
- The caret has moved. The entry field control must scroll to show the caret position.

CBN_LBSELECT

An item in the list box control has been selected.

CBN_LBSCROLL

The list box is about to scroll.

CBN_SHOWLIST

The list box is about to be displayed.

CBN_ENTER

The user has depressed the ENTER key or double clicked (single clicked in the case of a drop-down list) on an item in the list box control.

WM_CONTROL (in Combination Boxes) Field - hwndcontrolspect

hwndcontrolspect ([HWND](#))

Combination (combo) window handle.

WM_CONTROL (in Combination Boxes) Return Value - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_CONTROL (in Combination Boxes) - Parameters

usid ([USHORT](#))

Control window identity.

usnotifycode ([USHORT](#))

Notify code.

CBN_EFCHANGE

The content of the entry field control has changed, and the change has been displayed on the screen.

CBN_MEMERROR

The entry field control cannot allocate the storage necessary to accommodate window text of the length implied by

the [EM_SETTEXTLIMIT](#) message.

CBN_EFSCROLL

The entry field control is about to scroll horizontally. This can happen in these circumstances:

- The application has issued a [WinScrollWindow](#) call.
- The content of the entry field control has changed.
- The caret has moved. The entry field control must scroll to show the caret position.

CBN_LBSELECT

An item in the list box control has been selected.

CBN_LBSCROLL

The list box is about to scroll.

CBN_SHOWLIST

The list box is about to be displayed.

CBN_ENTER

The user has depressed the ENTER key or double clicked (single clicked in the case of a drop-down list) on an item in the list box control.

hwndcontrolspect ([HWND](#))

Combination (combo) window handle.

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_CONTROL (in Combination Boxes) - Syntax

For the cause of this message, see [WM_CONTROL](#).

```
param1
    USHORT  usid           /* Control window identity. */
    USHORT  usnotifycode   /* Notify code. */

param2
    HWND     hwndcontrolspect /* Combination (combo) window handle. */
```

WM_CONTROL (in Combination Boxes) - Remarks

The entry field control window procedure generates this message and sends it to its owner, informing the owner of the event.

WM_CONTROL (in Combination Boxes) - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_CONTROL (in Combination Boxes) - Related Messages

Related Messages

- [WM_CONTROL](#)

WM_CONTROL (in Combination Boxes) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

Combo Box Control Window Messages

The combo box control uses most of the same messages as the entry field control and the list box control. In particular, the following messages are supported to achieve the functions of a combo box. These messages are explained in detail in the entry field control window messages and the list box control window messages sections.

[WM_SETWINDOWPARAMS \(in Entry Fields\)](#)

To set the text of the entry field.

[WM_QUERYWINDOWPARAMS \(in Entry Fields\)](#)

To obtain the text of the entry field.

[LM_QUERYITEMCOUNT](#)

To obtain the count of items in the list box control.

[LM_INSERTITEM](#)

To insert an item into the list box control.

[LM_SETTOPINDEX](#)

To scroll the list box control so that the specified item is at the top.

[LM_QUERYTOPINDEX](#)

To obtain the index of the item at the top of the list box control.

[LM_DELETEITEM](#)

To delete an item from the list box control. If necessary, this also changes the content of the entry field to the item at the top of the list box control.

[LM_SELECTITEM](#)

To select a specified item in the list box control. Also, this changes the content of the entry field to the item at the top of the list box control and, if the list box control is not visible, causes the list box control to 'dropdown' below the entry field control.

[LM_QUERYSELECTION](#)

To obtain the current selection in the list box control.

[LM_SETITEMTEXT](#)

To change the text of an item in the list box control. If necessary, this also changes the content of the entry field control.

[LM_QUERYITEMTEXT](#)

To obtain the text of an item in the list box control.

[LM_QUERYITEMTEXTLENGTH](#)

To obtain the length of the text of an item in the list box control.

LM_SEARCHSTRING

To obtain the index of an item in the list box control containing a specified string.

LM_DELETEALL

To delete all the items in the list box control.

WM_ENABLE

To enable the combo box control to respond to input.

EM_QUERYFIRSTCHAR

To obtain the character displayed at the left edge of the entry field control.

EM_SETFIRSTCHAR

To scroll the entry field control so that the specified character is displayed at the left edge of the entry field control.

EM_QUERYCHANGED

To obtain the changes to the entry field control.

EM_QUERYSEL

To obtain the current selection of the entry field control.

EM_SETSEL

To set the current selection of the entry field control.

EM_SETTEXTLIMIT

To set the maximum number of characters to be contained in the entry field control.

EM_CUT

To place the contents of the selection of the entry field control into the clipboard and then delete those contents from the entry field control.

EM_PASTE

To place the contents of the clipboard into the entry field control.

EM_COPY

To place the contents of the selection of the entry field control into the clipboard.

EM_CLEAR

To clear the current selection of the entry field control.

CBM_HILITE

CBM_HILITE Field - usHilite

usHilite (USHORT)

Highlighting indicator.

TRUE

Highlight the entry field control.

FALSE

Do not highlight the entry field control.

CBM_HILITE Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

CBM_HILITE Return Value - rc

rc (BOOL)
Changed indicator.

TRUE	The highlighting state of the entry field has been changed.
FALSE	The highlighting state of the entry field has not been changed.

CBM_HILITE - Parameters

usHilite (USHORT)
Highlighting indicator.

TRUE	Highlight the entry field control.
FALSE	Do not highlight the entry field control.

ulReserved (ULONG)
Reserved value, should be 0.

rc (BOOL)
Changed indicator.

TRUE	The highlighting state of the entry field has been changed.
FALSE	The highlighting state of the entry field has not been changed.

CBM_HILITE - Syntax

This message sets the highlighting state of the entry field control.

```
param1
    USHORT  usHilite    /* Highlighting indicator. */

param2
    ULONG   ulReserved  /* Reserved value, should be 0. */
```

CBM_HILITE - Remarks

The combo box control window procedure responds to this message by setting the highlighting state of the entry field control.

CBM_HILITE - Default Processing

WinDefWindowProc does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

CBM_HILITE - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CBM_ISLISTSHOWING

CBM_ISLISTSHOWING Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

CBM_ISLISTSHOWING Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

CBM_ISLISTSHOWING Return Value - rc

rc (BOOL)

Showing indicator.

TRUE

The list box control is showing.

FALSE

The list box control is not showing.

CBM_ISLISTSHOWING - Parameters

ulReserved (ULONG)

Reserved value, should be 0.

ulReserved (ULONG)

Reserved value, should be 0.

rc (BOOL)

Showing indicator.

TRUE

The list box control is showing.

FALSE

The list box control is not showing.

CBM_ISLISTSHOWING - Syntax

This message determines if the list box control is showing.

```
param1
    ULONG    ulReserved    /* Reserved value, should be 0. */
```

```
param2
    ULONG    ulReserved    /* Reserved value, should be 0. */
```

CBM_ISLISTSHOWING - Remarks

The combo box control window procedure responds to this message by indicating if the list box control is showing.

CBM_ISLISTSHOWING - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the

default value of FALSE.

CBM_ISLISTSHOWING - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

CBM_SHOWLIST

CBM_SHOWLIST Field - usShowing

usShowing (USHORT)	
Showing indicator.	
TRUE	
	Show the list box control.
FALSE	
	Do not show the list box control.

CBM_SHOWLIST Field - ulReserved

ulReserved (ULONG)	
Reserved value, should be 0.	

CBM_SHOWLIST Return Value - rc

rc (BOOL)	
Changed indicator.	
TRUE	
	The list box showing state has been changed.
FALSE	
	The list box showing state has not been changed.

CBM_SHOWLIST - Parameters

usShowing ([USHORT](#))

Showing indicator.

TRUE

Show the list box control.

FALSE

Do not show the list box control.

ulReserved ([ULONG](#))

Reserved value, should be 0.

rc ([BOOL](#))

Changed indicator.

TRUE

The list box showing state has been changed.

FALSE

The list box showing state has not been changed.

CBM_SHOWLIST - Syntax

This message sets the showing state of the list box control.

```
param1
    USHORT  usShowing /* Showing indicator. */

param2
    ULONG   ulReserved /* Reserved value, should be 0. */
```

CBM_SHOWLIST - Remarks

The combo box control window procedure responds to this message by setting the showing state of the list box control.

This message has no effect on a combo box control whose style is CBS_SIMPLE.

Hiding the list box control has no effect on the selection in the list box control. The selection in the list box control must be changed by the use of a [LM_SELECTITEM](#) message.

CBM_SHOWLIST - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the

default value of FALSE.

CBM_SHOWLIST - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)
-

Container Control Window Processing

This system-provided window procedure processes the actions on a container control (WC_CONTAINER).

Purpose

A container control is a visual component whose specific purpose is to hold objects. These objects, or container items, can be anything that either your application or a user might store in a container. Examples are executable programs, word processing files, graphics images, and database records.

Container item data is stored in [RECORDCORE](#) or [MINIRECORDCORE](#) data structures. Both the application and the container have access to the data stored in these records.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

The maximum number of records is limited by the amount of memory in the user's computer. The container control does not limit the number of records that a container can have.

The following list shows which types of data can be displayed for each container view. Refer to the description of the container control in the *OS/2 Programming Guide* for more information about the types of views.

View Types	Data
Icon view	Icons or bit maps with text strings beneath
Grid view	Icons or bit maps arranged in grid squares
Name view	Icons or bit maps with text strings to the right
Text view	Text strings
Tree view	Icons or bit maps, and text strings
Details view	Icons or bit maps, text strings, numbers, times, and dates.

Direct editing of container item text is supported in all views, including blank text fields.

The container control is designed according to the Common User Access (CUA) guidelines. For example, the CUA direct manipulation protocol is fully supported, enabling a user to visually drag an object in a container window and drop it on another object or container window. In addition, the container control supports CUA-defined selection types and techniques for selecting container items, as well as selection mechanisms, such as pointing devices and the keyboard, and multiple forms of emphasis. For a complete description of CUA containers, refer to the *SAA CUA Guide to User Interface Design* and to the *SAA CUA Advanced Interface Design Reference*.

The container control automatically provides or enables either horizontal or vertical scroll bars, or both, whenever all or part of one or more container items are not visible in a container window's client area.

Container Control Window Words

The container control reserves 4 bytes in its window words for application use. This memory can be accessed using the

[WinSetWindowULong](#) and [WinQueryWindowULong](#) functions at offset QWL_USER.

Container Control Styles and Selection Types

Containers are WC_CONTAINER class windows that have the following CCS_container styles and selection types. Container control styles and selection types are specified when the container control is created.

Container Control Styles

The following list defines container style bits that your application can use. These style bits must be set by your application.

CCS_AUTOPOSITION

Automatic positioning, which causes container items displayed in the icon view to be arranged when any of the following occur:

- The window size changes
- Container items are inserted, removed, sorted, invalidated, or filtered
- The font or font size changes
- The window title text changes.

In all of these cases, container items are arranged the same as when the CM_ARRANGE message is sent. The CCS_AUTOPOSITION style bit is valid only when it is used with the icon view (CV_ICON).

CCS_MINIRECORDCORE

A record style bit that causes the container to interpret all container records as being smaller than they would otherwise be. If a [CM_ALLOCRECORD](#) message is received, all records are interpreted and allocated according to the information in the [MINIRECORDCORE](#) data structure instead of the [RECORDCORE](#) data structure, which is used if this style bit is not specified.

CCS_READONLY

A read-only style bit for an entire container, which prevents a user from editing any of the text in a container window. If you do not set this style bit, a user can edit any of the text in a container window unless you set the following read-only attributes in the appropriate data structures:

CA_TITLEREADONLY

Sets the container title to read-only. This is an attribute of the [CNRINFO](#) data structure's *fWindowAttr* field.

CRA_RECORDREADONLY

Sets text fields in records to read-only. This is an attribute of the [RECORDCORE](#) and [MINIRECORDCORE](#) data structures' *fRecordAttr* field.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, the [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

CFA_FIREADONLY

Sets column data to read-only. This is an attribute of the [FIELDINFO](#) data structure's *fData* field.

CFA_FITITLEREADONLY

Sets column headings to read-only. This is an attribute of the [FIELDINFO](#) data structure's *fTitle* field.

CCS_VERIFYPOINTERS

A pointer verification style bit, which verifies that the application pointers are members of the container's linked list before they are used. If it is not set, the container does not verify the pointers.

Notes:

1. The CCS_VERIFYPOINTERS style bit does not verify the validity of a pointer. It only verifies whether a pointer is a member of a container's linked list.
2. After your code has been developed and tested, you may want to remove the CCS_VERIFYPOINTERS style bit in order to improve the container's performance. Otherwise, the container will attempt to verify all pointers, which will slow its response to actions that users perform.

Container Control Selection Types

If a selection type is not specified, single selection is the default. For the tree view, single selection is the only type supported. Refer to the description of the selection types in the *SAA CUA Advanced Interface Design Reference* for more information.

CCS_SINGLESEL

Single selection, which allows a user to select only one container item at a time. Each time a user selects a container item, the selection of any other container item is cancelled.

CCS_EXTENDSEL

Extended selection, which allows a user to select one or more container items. A user can select one item, a range of items, or multiple ranges of items.

CCS_MULTIPLESEL

Multiple selection, which allows a user to select zero or more container items.

Container Control Data

See the following for information on the container control data structures:

- [CDATE](#)
- [CNRDRAGINFO](#)
- [CNRDRAGINIT](#)
- [CNRDRAWITEMINFO](#)
- [CNREDITDATA](#)
- [CNRINFO](#)
- [CTIME](#)
- [FIELDINFO](#)
- [FIELDINFOINSERT](#)
- [GRIDINFO](#)
- [GRIDSQUARE](#)
- [MINIRECORDCORE](#)
- [NOTIFYDELTA](#)
- [NOTIFYRECORDEMPHASIS](#)
- [NOTIFYRECORDENTER](#)
- [NOTIFYSCROLL](#)
- [OWNERBACKGROUND](#)
- [QUERYRECFROMRECT](#)
- [QUERYRECORDRECT](#)
- [RECORDCORE](#)
- [RECORDINSERT](#)
- [SEARCHSTRING](#)
- [TREEITEMDESC.](#)

Container Control Notification Messages

These messages are initiated by the container control window to notify its owner of significant events.

WM_CONTROL (in Container Controls)

WM_CONTROL (in Container Controls) Field - id

id ([USHORT](#))
Container control ID.

WM_CONTROL (in Container Controls) Field - notifycode

notifycode ([USHORT](#))
Notify code.

The container control uses the following notification codes. For the complete description of the specified *notifycode*, see [Container Control Notification Codes](#).

- CN_BEGINEDIT**
Container text is about to be edited.
- CN_COLLAPSETREE**
A parent item was collapsed in the tree view.
- CN_CONTEXTMENU**
The container received a WM_CONTEXTMENU message.
- CN_DRAGAFTER**
The container received a [DM_DRAGOVER](#) message. The CN_DRAGAFTER notification code is sent only if either the CA_ORDEREDTARGETEMPH or CA_MIXEDTARGETEMPH attribute of the [CNRINFO](#) data structure is set and the current view is the name, text, or details view.
- CN_DRAGLEAVE**
The container received a [DM_DRAGLEAVE](#) message.
- CN_DRAGOVER**
The container received a [DM_DRAGOVER](#) message. The CN_DRAGOVER notification code is sent only if the CA_ORDEREDTARGETEMPH attribute of the [CNRINFO](#) data structure is not set or the current view is the icon view or tree view.
- CN_DROP**
The container received a [DM_DROP](#) message.
- CN_DROPNOTIFY**
The container received a [DM_DROPNOTIFY](#) message.
- CN_DROPHELP**
The container received a [DM_DROPHELP](#) message.
- CN_EMPHASIS**
A container record's attributes changed.
- CN_ENDEDIT**
Direct editing of container text has ended.
- CN_ENTER**
The Enter key is pressed while the container window has the focus, or the select button is double-clicked while the pointer is over the container window.
- CN_EXPANDTREE**
A parent item is expanded in the tree view.
- CN_GRIDRESIZED**
The grid was resized. This means there are more or fewer squares than previously, and all the squares are now marked CM_AVAIL.
- CN_HELP**
The container received a [WM_HELP](#) message.
- CN_INITDRAG**
The drag button was pressed and the pointer was moved while the pointer was over the container control.

CN_KILLFOCUS	The container is losing the focus.
CN_PICKUP	The container received a WM_PICKUP message.
CN_QUERYDELTA	Queries for more data when a user scrolls to a preset delta value.
CN_REALLOCPSZ	Container text is edited. This message is sent before the CN_ENDEDIT notification code is sent.
CN_SCROLL	The container window scrolled.
CN_SETFOCUS	The container is receiving the focus.

WM_CONTROL (in Container Controls) Field - notifyinfo

notifyinfo ([ULONG](#))
Notify code information.

For the definition of this parameter, see the description of the specified *notifycode* in [Container Control Notification Codes](#).

WM_CONTROL (in Container Controls) Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_CONTROL (in Container Controls) - Parameters

id ([USHORT](#))
Container control ID.

notifycode ([USHORT](#))
Notify code.

The container control uses the following notification codes. For the complete description of the specified *notifycode*, see [Container Control Notification Codes](#).

CN_BEGINEDIT	Container text is about to be edited.
CN_COLLAPSETREE	A parent item was collapsed in the tree view.
CN_CONTEXTMENU	The container received a WM_CONTEXTMENU message.

CN_DRAGAFTER

The container received a [DM_DRAGOVER](#) message. The CN_DRAGAFTER notification code is sent only if either the CA_ORDEREDTARGETEMPH or CA_MIXEDTARGETEMPH attribute of the [CNRINFO](#) data structure is set and the current view is the name, text, or details view.

CN_DRAGLEAVE

The container received a [DM_DRAGLEAVE](#) message.

CN_DRAGOVER

The container received a [DM_DRAGOVER](#) message. The CN_DRAGOVER notification code is sent only if the CA_ORDEREDTARGETEMPH attribute of the [CNRINFO](#) data structure is not set or the current view is the icon view or tree view.

CN_DROP

The container received a [DM_DROP](#) message.

CN_DROPNOTIFY

The container received a [DM_DROPNOTIFY](#) message.

CN_DROPHELP

The container received a [DM_DROPHELP](#) message.

CN_EMPHASIS

A container record's attributes changed.

CN_ENDEDIT

Direct editing of container text has ended.

CN_ENTER

The Enter key is pressed while the container window has the focus, or the select button is double-clicked while the pointer is over the container window.

CN_EXPANDTREE

A parent item is expanded in the tree view.

CN_GRIDRESIZED

The grid was resized. This means there are more or fewer squares than previously, and all the squares are now marked CM_AVAIL.

CN_HELP

The container received a [WM_HELP](#) message.

CN_INITDRAG

The drag button was pressed and the pointer was moved while the pointer was over the container control.

CN_KILLFOCUS

The container is losing the focus.

CN_PICKUP

The container received a [WM_PICKUP](#) message.

CN_QUERYDELTA

Queries for more data when a user scrolls to a preset delta value.

CN_REALLOCPSZ

Container text is edited. This message is sent before the CN_ENDEDIT notification code is sent.

CN_SCROLL

The container window scrolled.

CN_SETFOCUS

The container is receiving the focus.

notifyinfo (ULONG)

Notify code information.

For the definition of this parameter, see the description of the specified *notifycode* in [Container Control Notification Codes](#).

ulReserved (ULONG)

Reserved value, should be 0.

WM_CONTROL (in Container Controls) - Syntax

For the cause of this message, see [WM_CONTROL](#).

```
param1
    USHORT id          /* Container control ID. */
    USHORT notifycode  /* Notify code. */

param2
    ULONG notifyinfo   /* Notify code information. */
```

WM_CONTROL (in Container Controls) - Remarks

The container control window procedure generates this message and sends it to its owner, informing the owner of this event.

WM_CONTROL (in Container Controls) - Default Processing

For a description of the default processing, see [WM_CONTROL](#).

WM_CONTROL (in Container Controls) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_CONTROLPOINTER (in Container Controls)

WM_CONTROLPOINTER (in Container Controls) - Syntax

For the cause of this message, see [WM_CONTROLPOINTER](#).

For a description of the parameters, see [WM_CONTROLPOINTER](#).

WM_CONTROLPOINTER (in Container Controls) - Remarks

For the appropriate remarks, see [WM_CONTROLPOINTER](#).

WM_CONTROLPOINTER (in Container Controls) - Default Processing

For the default processing, see [WM_CONTROLPOINTER](#).

WM_CONTROLPOINTER (in Container Controls) - Topics

- Select an item:
- [Syntax](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_DRAWITEM (in Container Controls)

WM_DRAWITEM (in Container Controls) Field - id

id ([USHORT](#))
Container control ID.

WM_DRAWITEM (in Container Controls) Field - pOwnerItem

pOwnerItem ([POWNERITEM](#))
Pointer to an [OWNERITEM](#) data structure.

The following list defines the [OWNERITEM](#) data structure fields as they apply to the container control. See [OWNERITEM](#) for the default field values.

hwnd (HWND)

Handle of the window in which ownerdraw will occur. The following is a list of the window handles that can be specified for ownerdraw:

- The container window handle of the icon, name, text, and tree views
- The container title window handle
- The left or right window handles of the details view
- The left or right column heading windows of the details view.

hps (HPS)

Handle of the presentation space of the container window. For the details view that uses a split bar, the presentation space handle is either for the left or right window, depending upon the position of the column. If the details view does not have a split bar, the presentation space handle is for the left window.

fsState (ULONG)

Specifies emphasis flags. This state is not used by the container control because the application is responsible for drawing the emphasis states during ownerdraw.

fsAttribute (ULONG)

Attributes of the record as given in the *//RecordAttr* field in the RECORDCORE data structure.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

fsStateOld (ULONG)

Previous emphasis. This state is not used by the container control because the application is responsible for drawing the emphasis states during ownerdraw.

fsAttributeOld (ULONG)

Previous attribute. This state is not used by the container control because the application is responsible for drawing the emphasis states during ownerdraw.

rcItem (RECT)

This is the bounding rectangle into which the container item is drawn.

If the container item is an icon/text or bit-map/text pair, two WM_DRAWITEM messages are sent to the application. The first WM_DRAWITEM message contains the rectangle bounding the icon or bit map and the second contains the rectangle bounding the text.

If the container item contains only text, or only an icon or bit map, only one WM_DRAWITEM message is sent. However, if the current view is the tree icon or tree text view and if the item is a parent item, the application will receive an additional WM_DRAWITEM (in Container Controls) message. The additional message is for the icon or bit map that indicates whether the parent item is expanded or collapsed.

If the current view is the details view and the CFA_OWNER attribute is set, the rectangle's size is equal to the width of the column and the height of the tallest field in the container item. CFA_OWNER is an attribute of the FIELDINFO data structure's *//Data* field.

idItem (ULONG)

Identifies the item being drawn. It can be one of the following:

- CMA_CNRTITLE
- CMA_ICON
- CMA_TEXT
- CMA_TREEICON.

This field is not used for the details view and is set to 0.

hItem (CNRDRAWITEMINFO)

Pointer to a CNRDRAWITEMINFO structure.

WM_DRAWITEM (in Container Controls) Return Value - rc

rc ([BOOL](#))

Item-drawn indicator.

TRUE

The owner draws the item, and so the container control does not draw it.

FALSE

If the owner does not draw the item, the owner returns this value and the container control draws the item.

WM_DRAWITEM (in Container Controls) - Parameters

id ([USHORT](#))

Container control ID.

pOwnerItem ([POWNERITEM](#))

Pointer to an [OWNERITEM](#) data structure.

The following list defines the [OWNERITEM](#) data structure fields as they apply to the container control. See [OWNERITEM](#) for the default field values.

hwnd ([HWND](#))

Handle of the window in which ownerdraw will occur. The following is a list of the window handles that can be specified for ownerdraw:

- The container window handle of the icon, name, text, and tree views
- The container title window handle
- The left or right window handles of the details view
- The left or right column heading windows of the details view.

hps ([HPS](#))

Handle of the presentation space of the container window. For the details view that uses a split bar, the presentation space handle is either for the left or right window, depending upon the position of the column. If the details view does not have a split bar, the presentation space handle is for the left window.

fsState ([ULONG](#))

Specifies emphasis flags. This state is not used by the container control because the application is responsible for drawing the emphasis states during ownerdraw.

fsAttribute ([ULONG](#))

Attributes of the record as given in the *flRecordAttr* field in the [RECORDCORE](#) data structure.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

fsStateOld ([ULONG](#))

Previous emphasis. This state is not used by the container control because the application is responsible for drawing the emphasis states during ownerdraw.

fsAttributeOld ([ULONG](#))

Previous attribute. This state is not used by the container control because the application is responsible for drawing the emphasis states during ownerdraw.

rcItem ([RECTL](#))

This is the bounding rectangle into which the container item is drawn.

If the container item is an icon/text or bit-map/text pair, two WM_DRAWITEM messages are sent to the application. The first WM_DRAWITEM message contains the rectangle bounding the icon or bit map and the second contains the rectangle bounding the text.

If the container item contains only text, or only an icon or bit map, only one WM_DRAWITEM message is sent. However, if the current view is the tree icon or tree text view and if the item is a parent item, the application will receive an additional WM_DRAWITEM (in Container Controls) message. The additional message is for the icon or bit map that indicates whether the parent item is expanded or collapsed.

If the current view is the details view and the CFA_OWNER attribute is set, the rectangle's size is equal to the

width of the column and the height of the tallest field in the container item. CFA_OWNER is an attribute of the [FIELDINFO](#) data structure's *flData* field.

idItem ([ULONG](#))

Identifies the item being drawn. It can be one of the following:

- CMA_CNRTITLE
- CMA_ICON
- CMA_TEXT
- CMA_TREEICON.

This field is not used for the details view and is set to 0.

hItem ([CNRDRAWITEMINFO](#))

Pointer to a [CNRDRAWITEMINFO](#) structure.

rc ([BOOL](#))

Item-drawn indicator.

TRUE

The owner draws the item, and so the container control does not draw it.

FALSE

If the owner does not draw the item, the owner returns this value and the container control draws the item.

WM_DRAWITEM (in Container Controls) - Syntax

For the cause of this message, see [WM_DRAWITEM](#).

```
param1
    USHORT      id          /* Container control ID. */

param2
    POWNERITEM  pOwnerItem /* Pointer to an OWNERITEM data structure. */
```

WM_DRAWITEM (in Container Controls) - Remarks

CA_OWNERDRAW is an attribute of the [CNRINFO](#) data structure's *flWindowAttr* field.

The container control window procedure generates this message and sends it to the owner of the container control to offer the owner the opportunity to draw that item.

WM_DRAWITEM (in Container Controls) - Default Processing

For a description of the default processing, see [WM_DRAWITEM](#).

WM_DRAWITEM (in Container Controls) - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

Container Control Notification Codes

The following [WM_CONTROL](#) (in [Container Controls](#)) notification codes are sent by the container control to its owner.

CN_BEGINEDIT

CN_BEGINEDIT Field - id

id ([USHORT](#))
Container control ID.

CN_BEGINEDIT Field - CN_BEGINEDIT

CN_BEGINEDIT ([USHORT](#))
Notification code.

CN_BEGINEDIT Field - pCnrEditData

pCnrEditData ([PCNREDITDATA](#))
Pointer to the [CNREDITDATA](#) structure.

CN_BEGINEDIT Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_BEGINEDIT - Parameters

id ([USHORT](#))
Container control ID.

CN_BEGINEDIT ([USHORT](#))
Notification code.

pCnrEditData ([PCNREDITDATA](#))
Pointer to the [CNREDITDATA](#) structure.

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_BEGINEDIT - Syntax

The container control sends the [WM_CONTROL](#) (in [Container Controls](#)) message with the CN_BEGINEDIT notification code to its owner whenever container text is about to be edited.

```
param1
    USHORT    id          /* Container control ID. */
    USHORT    CN_BEGINEDIT /* Notification code. */

param2
    PCNREDITDATA pCnrEditData /* Pointer to the CNREDITDATA structure. */
```

CN_BEGINEDIT - Remarks

The CN_BEGINEDIT notification code is sent when direct editing of container text begins.

Warning: Once your application receives the CN_BEGINEDIT notification code, it must not send any messages to the container until it receives the [CN_ENDEDIT](#) notification code, which indicates that direct editing of container text has ended. If any messages are sent to the container before your application receives the [CN_ENDEDIT](#) notification code, the results of direct editing are unpredictable.

CN_BEGINEDIT - Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_BEGINEDIT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

CN_COLLAPSETREE

CN_COLLAPSETREE Field - id

id ([USHORT](#))
Container control ID.

CN_COLLAPSETREE Field - CN_COLLAPSETREE

CN_COLLAPSETREE ([USHORT](#))
Notification code.

CN_COLLAPSETREE Field - pRecord

pRecord ([PRECORDCORE](#))
Pointer to the record that was collapsed.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

CN_COLLAPSETREE Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_COLLAPSETREE - Parameters

id ([USHORT](#))
Container control ID.

CN_COLLAPSETREE ([USHORT](#))
Notification code.

pRecord ([PRECORDCORE](#))
Pointer to the record that was collapsed.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_COLLAPSETREE - Syntax

The container control sends the [WM_CONTROL](#) (in [Container Controls](#)) message with the CN_COLLAPSETREE notification code to its owner whenever the container collapses a parent item in the tree view.

```
param1
    USHORT      id          /* Container control ID. */
    USHORT      CN_COLLAPSETREE /* Notification code. */

param2
    PRECORDCORE pRecord     /* Pointer to the record that was collapsed. */
```

CN_COLLAPSETREE - Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_COLLAPSETREE - Topics

Select an item:
[Syntax](#)

- Parameters
- Returns
- Default Processing
- Glossary

CN_CONTEXTMENU

CN_CONTEXTMENU Field - id

id (USHORT)
Container control ID.

CN_CONTEXTMENU Field - CN_CONTEXTMENU

CN_CONTEXTMENU (**USHORT**)
Notification code.

CN_CONTEXTMENU Field - pRecord

pRecord (PRECORDCORE)
Pointer to the RECORDCORE structure.

If the user is using a pointing device, this **RECORDCORE** structure is the structure that the pointing device pointer is over. If the pointing device pointer is over white space, this field is NULL.

If the user is using the keyboard, this **RECORDCORE** structure is the structure that has the selection cursor.

CN_CONTEXTMENU Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

CN_CONTEXTMENU - Parameters

id ([USHORT](#))
Container control ID.

CN_CONTEXTMENU ([USHORT](#))
Notification code.

pRecord ([PRECORDCORE](#))
Pointer to the [RECORDCORE](#) structure.

If the user is using a pointing device, this [RECORDCORE](#) structure is the structure that the pointing device pointer is over. If the pointing device pointer is over white space, this field is NULL.

If the user is using the keyboard, this [RECORDCORE](#) structure is the structure that has the selection cursor.

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_CONTEXTMENU - Syntax

The container control sends the [WM_CONTROL \(in Container Controls\)](#) message with the CN_CONTEXTMENU notification code to its owner when the container receives a [WM_CONTEXTMENU](#) message.

```
param1
    USHORT      id          /* Container control ID. */
    USHORT      CN_CONTEXTMENU /* Notification code. */

param2
    PRECORDCORE pRecord     /* Pointer to the RECORDCORE structure. */
```

CN_CONTEXTMENU - Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_CONTEXTMENU - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Glossary](#)

CN_DRAGAFTER

CN_DRAGAFTER Field - id

id ([USHORT](#))
Container control ID.

CN_DRAGAFTER Field - CN_DRAGAFTER

CN_DRAGAFTER ([USHORT](#))
Notification code.

CN_DRAGAFTER Field - pCnrDragInfo

pCnrDragInfo ([PCNRDRAGINFO](#))
Pointer to a [CNRDRAGINFO](#) structure.

CN_DRAGAFTER Field - usDrop

usDrop ([USHORT](#))
Drop indicator.

DOR_DROP

The record can be dropped. The drop will not occur unless DOR_DROP is returned. When this response is returned, the container control applies ordered target emphasis to the target record.

DOR_NODROP

The record is acceptable and the current operation is supported by the target, but the record cannot be dropped in the current location. For example, the container control returns DOR_NODROP if the record being dragged is positioned over another record on which it cannot be dropped.

If the container returns DOR_NODROP, the [DM_DRAGOVER](#) message will continue to be sent to it when the user does any of the following:

- Moves the pointer
- Presses a keyboard key
- Moves the pointer out of and back into the container window.

DOR_NODROPOP

The record is acceptable, but the target does not support the current operation. This response implies that the drop may be valid if the drag operation changes. For example, if the default operation is copy and the target does not support this operation, the drop may become valid if the user presses a keyboard augmentation key to change to a different operation, such as move.

If the container returns DOR_NODROPOP, no further [DM_DRAGOVER](#) messages are sent until the user does any of the following:

- Presses a keyboard key
- Moves the pointer out of and back into the container window.

DOR_NEVERDROP

The record cannot be dropped. Ordered target emphasis is not drawn. If the container returns DOR_NEVERDROP, no further [DM_DRAGOVER](#) messages are sent until the user drags the record outside of and back into the container window.

CN_DRAGAFTER Field - usDefaultOp

usDefaultOp ([USHORT](#))

Default operation.

Target-defined default operation.

DO_COPY

Operation is a copy.

DO_DEFAULT

Operation is the default drag operation. No modifier keys are pressed.

DO_LINK

Operation is a link.

DO_MOVE

Operation is a move.

DO_UNKNOWN

Operation is application-defined.

CN_DRAGAFTER - Parameters

id ([USHORT](#))

Container control ID.

CN_DRAGAFTER ([USHORT](#))

Notification code.

pCnrDragInfo ([PCNRDRAGINFO](#))

Pointer to a [CNRDRAGINFO](#) structure.

usDrop ([USHORT](#))

Drop indicator.

DOR_DROP

The record can be dropped. The drop will not occur unless DOR_DROP is returned. When this response is returned, the container control applies ordered target emphasis to the target record.

DOR_NODROP

The record is acceptable and the current operation is supported by the target, but the record cannot be dropped in the current location. For example, the container control returns DOR_NODROP if the record being dragged is positioned over another record on which it cannot be dropped.

If the container returns DOR_NODROP, the [DM_DRAGOVER](#) message will continue to be sent to it when the user

does any of the following:

- Moves the pointer
- Presses a keyboard key
- Moves the pointer out of and back into the container window.

DOR_NODROPOP

The record is acceptable, but the target does not support the current operation. This response implies that the drop may be valid if the drag operation changes. For example, if the default operation is copy and the target does not support this operation, the drop may become valid if the user presses a keyboard augmentation key to change to a different operation, such as move.

If the container returns DOR_NODROPOP, no further [DM_DRAGOVER](#) messages are sent until the user does any of the following:

- Presses a keyboard key
- Moves the pointer out of and back into the container window.

DOR_NEVERDROP

The record cannot be dropped. Ordered target emphasis is not drawn. If the container returns DOR_NEVERDROP, no further [DM_DRAGOVER](#) messages are sent until the user drags the record outside of and back into the container window.

usDefaultOp (USHORT)

Default operation.

Target-defined default operation.

DO_COPY

Operation is a copy.

DO_DEFAULT

Operation is the default drag operation. No modifier keys are pressed.

DO_LINK

Operation is a link.

DO_MOVE

Operation is a move.

DO_UNKNOWN

Operation is application-defined.

CN_DRAGAFTER - Syntax

The container control sends a [WM_CONTROL \(in Container Controls\)](#) message with the CN_DRAGAFTER notification code to its owner whenever the container receives a [DM_DRAGOVER](#) message. The CN_DRAGAFTER notification code is sent only if the CA_ORDEREDTARGETEMPHASIS or CA_MIXEDTARGETEMPHASIS attribute of the [CNRINFO](#) data structure is set and the current view is the name, text, or details view.

```
param1
    USHORT    id          /* Container control ID. */
    USHORT    CN_DRAGAFTER /* Notification code. */

param2
    PCNRDRAGINFO pCnrDragInfo /* Pointer to a CNRDRAGINFO structure. */

returns
    USHORT    usDrop      /* Drop indicator. */
    USHORT    usDefaultOp /* Default operation. */
```

CN_DRAGAFTER - Remarks

The container control draws ordered target emphasis of container records. The target emphasis provided by the container control is a black line that is drawn below the target record. Therefore, it is not necessary for the application to draw any emphasis for the container when it receives this notification code.

If the container returns anything except DOR_DROP, the target emphasis is automatically changed to a symbol that indicates no drop is allowed. This gives the user a visual cue that a drop cannot occur. The symbol reverts to the black line when the container returns a DOR_DROP reply.

The CN_DRAGAFTER notification code is sent only for the details, name, and text views when the CA_ORDEREDTARGETEMPHASIS or CA_MIXEDTARGETEMPHASIS attribute of the CNRINFO data structure is set. If this attribute is not set, the CN_DRAGOVER notification code is sent.

CN_DRAGAFTER - Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_DRAGAFTER - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

CN_DRAGLEAVE

CN_DRAGLEAVE Field - id

id (USHORT)
Container control ID.

CN_DRAGLEAVE Field - CN_DRAGLEAVE

CN_DRAGLEAVE ([USHORT](#))
Notification code.

CN_DRAGLEAVE Field - pCnrDragInfo

pCnrDragInfo ([PCNRDRAGINFO](#))
Pointer to a [CNRDRAGINFO](#) structure.

CN_DRAGLEAVE Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_DRAGLEAVE - Parameters

id ([USHORT](#))
Container control ID.

CN_DRAGLEAVE ([USHORT](#))
Notification code.

pCnrDragInfo ([PCNRDRAGINFO](#))
Pointer to a [CNRDRAGINFO](#) structure.

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_DRAGLEAVE - Syntax

The container control sends a [WM_CONTROL \(in Container Controls\)](#) message with the CN_DRAGLEAVE notification code to its owner when the container receives a [DM_DRAGLEAVE](#) message.

```
param1
    USHORT      id          /* Container control ID. */
    USHORT      CN_DRAGLEAVE /* Notification code. */

param2
    PCNRDRAGINFO pCnrDragInfo /* Pointer to a CNRDRAGINFO structure. */
```

CN_DRAGLEAVE - Remarks

This notification code is sent to the owner of the container control in response to a [DM_DRAGLEAVE](#) message. It informs the owner that one of the following has occurred:

- A container record was being dragged over the container and has left the container's boundaries.
- The drag ended when help was requested or a user pressed the Esc key while the container record was over the container.

CN_DRAGLEAVE - Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_DRAGLEAVE - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CN_DRAGOVER

CN_DRAGOVER Field - id

id ([USHORT](#))
Container control ID.

CN_DRAGOVER Field - CN_DRAGOVER

CN_DRAGOVER ([USHORT](#))
Notification code.

CN_DRAGOVER Field - pCnrDragInfo

pCnrDragInfo ([PCNRDRAGINFO](#))
Pointer to a [CNRDRAGINFO](#) structure.

CN_DRAGOVER Field - usDrop

usDrop ([USHORT](#))
Drop indicator.

DOR_DROP The record can be dropped. When this response is returned, the container control applies target emphasis.

DOR_NODROP The record is acceptable and the current operation is supported by the target, but the record cannot be dropped in the current location. For example, the container control returns DOR_DROP if the record being dragged is positioned over another record on which it cannot be dropped.

If the container returns DOR_NODROP, the [DM_DRAGOVER](#) message will continue to be sent to it when the user does any of the following:

- Moves the pointer
- Presses a keyboard key
- Moves the pointer out of and back into the container window.

DOR_NODROPOP The record is acceptable, but the target does not support the current operation. This response implies that the drop may be valid if the drag operation changes. For example, if the default operation is copy and the target does not support this operation, the drop may become valid if the user presses a keyboard augmentation key to change to a different operation, such as move.

If the container returns DOR_NODROPOP, no further [DM_DRAGOVER](#) messages are sent until the user does any of the following:

- Presses a keyboard key
- Moves the pointer out of and back into the container window.

DOR_NEVERDROP The record cannot be dropped. Target emphasis is not drawn. If the container returns DOR_NEVERDROP, no further [DM_DRAGOVER](#) messages are sent until the user drags the record outside of and back into the container window.

CN_DRAGOVER Field - usDefaultOp

usDefaultOp ([USHORT](#))
Default operation.

Target-defined default operation.

DO_COPY Operation is a copy.

DO_DEFAULT

DO_LINK	Operation is the default drag operation. No modifier keys are pressed.
DO_MOVE	Operation is a link.
DO_UNKNOWN	Operation is a move.
	Operation is application-defined.

CN_DRAGOVER - Parameters

id ([USHORT](#))
Container control ID.

CN_DRAGOVER ([USHORT](#))
Notification code.

pCnrDragInfo ([PCNRDRAGINFO](#))
Pointer to a [CNRDRAGINFO](#) structure.

usDrop ([USHORT](#))
Drop indicator.

DOR_DROP
The record can be dropped. When this response is returned, the container control applies target emphasis.

DOR_NODROP
The record is acceptable and the current operation is supported by the target, but the record cannot be dropped in the current location. For example, the container control returns DOR_DROP if the record being dragged is positioned over another record on which it cannot be dropped.

If the container returns DOR_NODROP, the [DM_DRAGOVER](#) message will continue to be sent to it when the user does any of the following:

- Moves the pointer
- Presses a keyboard key
- Moves the pointer out of and back into the container window.

DOR_NODROPOP
The record is acceptable, but the target does not support the current operation. This response implies that the drop may be valid if the drag operation changes. For example, if the default operation is copy and the target does not support this operation, the drop may become valid if the user presses a keyboard augmentation key to change to a different operation, such as move.

If the container returns DOR_NODROPOP, no further [DM_DRAGOVER](#) messages are sent until the user does any of the following:

- Presses a keyboard key
- Moves the pointer out of and back into the container window.

DOR_NEVERDROP
The record cannot be dropped. Target emphasis is not drawn. If the container returns DOR_NEVERDROP, no further [DM_DRAGOVER](#) messages are sent until the user drags the record outside of and back into the container window.

usDefaultOp ([USHORT](#))
Default operation.

Target-defined default operation.

DO_COPY
Operation is a copy.

DO_DEFAULT
Operation is the default drag operation. No modifier keys are pressed.

DO_LINK
Operation is a link.

DO_MOVE Operation is a move.
DO_UNKNOWN Operation is application-defined.

CN_DRAGOVER - Syntax

The container control sends a [WM_CONTROL \(in Container Controls\)](#) message with the CN_DRAGOVER notification code to its owner when the container receives a [DM_DRAGOVER](#) message. The CN_DRAGOVER notification code is sent only if the CA_ORDEREDTARGETEMPH attribute of the [CNRINFO](#) data structure is not set or the current view is the icon view or tree view.

```
param1
    USHORT      id          /* Container control ID. */
    USHORT      CN_DRAGOVER /* Notification code. */

param2
    PCNRDRAGINFO pCnrDragInfo /* Pointer to a CNRDRAGINFO structure. */

returns
    USHORT      usDrop      /* Drop indicator. */
    USHORT      usDefaultOp /* Default operation. */
```

CN_DRAGOVER - Remarks

This notification code shows where direct manipulation is occurring by applying target emphasis to indicate whether an item that is being dragged over the container can be dropped. It is not necessary for the application to draw any target emphasis for the container when it receives this notification code.

If the pointer is over a container record and the item that is being dragged can be dropped on that record, the container draws a black rectangle around the target record. If the pointer is over white space and the item that is being dragged can be dropped on the white space, the container draws a black border around the edge of the client area.

If the container returns anything except DOR_DROP, the target emphasis is automatically changed to a symbol that indicates no drop is allowed. This gives the user a visual cue that a drop cannot occur. The symbol reverts to the black rectangle or black border when the container returns a DOR_DROP reply.

The CN_DRAGOVER notification code is sent only for the icon and tree views, or when the CA_ORDEREDTARGETEMPH attribute of the [CNRINFO](#) data structure is not set. If this attribute is set and the current view is the name, text, or details view, the [CN_DRAGAFTER](#) notification code is sent.

The return parameter is reserved.

CN_DRAGOVER - Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_DRAGOVER - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CN_DROP

CN_DROP Field - id

id ([USHORT](#))
Container control ID.

CN_DROP Field - CN_DROP

CN_DROP ([USHORT](#))
Notification code.

CN_DROP Field - pCnrDragInfo

pCnrDragInfo ([PCNRDRAGINFO](#))
Pointer to a [CNRDRAGINFO](#) structure.

CN_DROP Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_DROP - Parameters

id ([USHORT](#))
Container control ID.

CN_DROP ([USHORT](#))
Notification code.

pCnrDragInfo ([PCNRDRAGINFO](#))
Pointer to a [CNRDRAGINFO](#) structure.

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_DROP - Syntax

The container control sends a [WM_CONTROL](#) (in [Container Controls](#)) message with the CN_DROP notification code to its owner when the container receives a [DM_DROP](#) message.

```
param1
    USHORT      id          /* Container control ID. */
    USHORT      CN_DROP     /* Notification code. */

param2
    PCNRDRAGINFO pCnrDragInfo /* Pointer to a CNRDRAGINFO structure. */
```

CN_DROP - Remarks

This notification code is sent to the container's owner when dragged container records are dropped over the container window.

CN_DROP - Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_DROP - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CN_DROPNOTIFY

CN_DROPNOTIFY Field - id

id ([USHORT](#))
Container control ID.

CN_DROPNOTIFY Field - CN_DROPNOTIFY

CN_DROPNOTIFY ([USHORT](#))
Notification code.

CN_DROPNOTIFY Field - pCnrLazyDragInfo

pCnrLazyDragInfo ([PCNRLAZYDRAGINFO](#))
Pointer to the [CNRLAZYDRAGINFO](#) structure.

This structure contains information about the [DRAGINFO](#), the [RECORDCORE](#) that was dropped on, and the window handle of the target window.

CN_DROPNOTIFY Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, must be 0.

CN_DROPNOTIFY - Parameters

id ([USHORT](#))
Container control ID.

CN_DROPNOTIFY ([USHORT](#))

Notification code.

pCnrLazyDragInfo ([PCNRLAZYDRAGINFO](#))

Pointer to the [CNRLAZYDRAGINFO](#) structure.

This structure contains information about the [DRAGINFO](#), the [RECORDCORE](#) that was dropped on, and the window handle of the target window.

ulReserved ([ULONG](#))

Reserved value, must be 0.

CN_DROPNOTIFY - Syntax

The container control sends a [WM_CONTROL \(in Container Controls\)](#) message with the CN_DROPNOTIFY notification code to its owner when a pickup set is dropped over the container.

```
param1
    USHORT          id          /* Container control ID. */
    USHORT          CN_DROPNOTIFY /* Notification code. */

param2
    PCNRLAZYDRAGINFO pCnrLazyDragInfo /* Pointer to the CNRLAZYDRAGINFO structure. */
```

CN_DROPNOTIFY - Remarks

This notification code is sent to the owner of the container when a lazy drag set is dropped over the container. (The container control receives a [DM_DROP](#) message.)

CN_DROPNOTIFY - Default Processing

The default window procedure does not expect to receive this notification and so takes no action on it other than returning 0.

CN_DROPNOTIFY - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

CN_DROPHELP

CN_DROPHELP Field - id

id (USHORT)
Container control ID.

CN_DROPHELP Field - CN_DROPHELP

CN_DROPHELP (USHORT)
Notification code.

CN_DROPHELP Field - pCnrDragInfo

pCnrDragInfo (PCNRDRAGINFO)
Pointer to a CNRDRAGINFO structure.

CN_DROPHELP Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

CN_DROPHELP - Parameters

id (USHORT)
Container control ID.

CN_DROPHELP (USHORT)
Notification code.

pCnrDragInfo (PCNRDRAGINFO)
Pointer to a CNRDRAGINFO structure.

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_DROPHELP - Syntax

The container control sends a [WM_CONTROL \(in Container Controls\)](#) message with the CN_DROPHELP notification code to its owner when the container receives a [DM_DROPHELP](#) message.

```
param1
    USHORT      id          /* Container control ID. */
    USHORT      CN_DROPHELP /* Notification code. */

param2
    PCNRDRAGINFO pCnrDragInfo /* Pointer to a CNRDRAGINFO structure. */
```

CN_DROPHELP - Remarks

This notification code is sent to the container's owner when help for direct manipulation is requested over the container window.

CN_DROPHELP - Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_DROPHELP - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

CN_EMPHASIS

CN_EMPHASIS Field - id

id (USHORT)
Container control ID.

CN_EMPHASIS Field - CN_EMPHASIS

CN_EMPHASIS (USHORT)
Notification code.

CN_EMPHASIS Field - pNotifyRecordEmphasis

pNotifyRecordEmphasis (PNOTIFYRECORDEMPHASIS)
Pointer to the NOTIFYRECORDEMPHASIS structure.

CN_EMPHASIS Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

CN_EMPHASIS - Parameters

id (USHORT)
Container control ID.

CN_EMPHASIS (USHORT)
Notification code.

pNotifyRecordEmphasis (PNOTIFYRECORDEMPHASIS)
Pointer to the NOTIFYRECORDEMPHASIS structure.

ulReserved (ULONG)
Reserved value, should be 0.

CN_EMPHASIS - Syntax

The container control sends a [WM_CONTROL \(in Container Controls\)](#) message with the CN_EMPHASIS notification code to its owner whenever a container record's attributes change.

```
param1
    USHORT          id          /* Container control ID. */
    USHORT          CN_EMPHASIS /* Notification code. */

param2
    NOTIFYRECORDEMPHASIS pNotifyRecordEmphasis /* Pointer to the NOTIFYRECORDEMPHASIS structure. */
```

CN_EMPHASIS - Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_EMPHASIS - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Default Processing](#)
 - [Glossary](#)

CN_ENDEDIT

CN_ENDEDIT Field - id

id ([USHORT](#))
Container control ID.

CN_ENDEDIT Field - CN_ENDEDIT

CN_ENDEDIT ([USHORT](#))
Notification code.

CN_ENDEDIT Field - pCnrEditData

pCnrEditData ([PCNREDITDATA](#))
Pointer to the [CNREDITDATA](#) structure.

CN_ENDEDIT Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_ENDEDIT - Parameters

- id** ([USHORT](#))
Container control ID.
- CN_ENDEDIT** ([USHORT](#))
Notification code.
- pCnrEditData** ([PCNREDITDATA](#))
Pointer to the [CNREDITDATA](#) structure.
- ulReserved** ([ULONG](#))
Reserved value, should be 0.
-

CN_ENDEDIT - Syntax

The container control sends a [WM_CONTROL \(in Container Controls\)](#) message with the CN_ENDEDIT notification code to its owner whenever direct editing of container text has ended.

```
param1
    USHORT      id          /* Container control ID. */
    USHORT      CN_ENDEDIT  /* Notification code. */

param2
    PCNREDITDATA pCnrEditData /* Pointer to the CNREDITDATA structure. */
```

CN_ENDEDIT - Remarks

Direct editing of container text is completed. Any changes made to the text are saved when a user presses the select button outside the window that contains the multiple-line entry (MLE) field used to edit text in a container. However, a user can end the direct editing of text without saving any changes to the text by doing any of the following:

- Pressing the Esc key
- Dragging the container item that is being edited
- Pressing the Alt key and the select button before direct editing of container text has ended
- Scrolling the container window.

The CN_ENDEDIT notification code is sent to the application in each of these cases.

CN_ENDEDIT - Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_ENDEDIT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

CN_ENTER

CN_ENTER Field - id

id ([USHORT](#))
Container control ID.

CN_ENTER Field - CN_ENTER

CN_ENTER ([USHORT](#))
Notification code.

CN_ENTER Field - pNotifyRecordEnter

pNotifyRecordEnter ([PNOTIFYRECORDENTER](#))
Pointer to the [NOTIFYRECORDENTER](#) structure.

CN_ENTER Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_ENTER - Parameters

id ([USHORT](#))
Container control ID.

CN_ENTER ([USHORT](#))
Notification code.

pNotifyRecordEnter ([PNOTIFYRECORDENTER](#))
Pointer to the [NOTIFYRECORDENTER](#) structure.

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_ENTER - Syntax

The container control sends a [WM_CONTROL \(in Container Controls\)](#) message with the CN_ENTER notification code to its owner when either of the following occurs:

- The Enter key is pressed while the container window has the focus
- The select button is double-clicked while the pointer is over the container window.

```
param1
    USHORT          id          /* Container control ID. */
    USHORT          CN_ENTER    /* Notification code. */

param2
    PNOTIFYRECORDENTER pNotifyRecordEnter /* Pointer to the NOTIFYRECORDENTER structure. */
```

CN_ENTER - Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_ENTER - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Glossary](#)

CN_EXPANDTREE

CN_EXPANDTREE Field - id

id ([USHORT](#))
Container control ID.

CN_EXPANDTREE Field - CN_EXPANDTREE

CN_EXPANDTREE ([USHORT](#))
Notification code.

CN_EXPANDTREE Field - pRecord

pRecord ([PRECORDCORE](#))
Pointer to the record that was expanded.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

CN_EXPANDTREE Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_EXPANDTREE - Parameters

id ([USHORT](#))
Container control ID.

CN_EXPANDTREE ([USHORT](#))
Notification code.

pRecord ([PRECORDCORE](#))
Pointer to the record that was expanded.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_EXPANDTREE - Syntax

The container control sends the [WM_CONTROL \(in Container Controls\)](#) message with the CN_EXPANDTREE notification code to its owner whenever the container expands a parent item in the tree view.

```
param1
    USHORT      id          /* Container control ID. */
    USHORT      CN_EXPANDTREE /* Notification code. */

param2
    PRECORDCORE pRecord     /* Pointer to the record that was expanded. */
```

CN_EXPANDTREE - Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_EXPANDTREE - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Glossary](#)

CN_GRIDRESIZED

CN_GRIDRESIZED Field - id

id ([USHORT](#))
Container control ID.

CN_GRIDRESIZED Field - CN_GRIDRESIZED

CN_GRIDRESIZED ([USHORT](#))
Notification code.

CN_GRIDRESIZED Field - sGridRows

sGridRows ([SHORT](#))
New number of rows.

CN_GRIDRESIZED Field - sGridCols

sGridCols ([SHORT](#))
New number of columns.

CN_GRIDRESIZED Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

CN_GRIDRESIZED - Parameters

id (USHORT)
Container control ID.

CN_GRIDRESIZED (USHORT)
Notification code.

sGridRows (SHORT)
New number of rows.

sGridCols (SHORT)
New number of columns.

ulReserved (ULONG)
Reserved value, should be 0.

CN_GRIDRESIZED - Syntax

This WM_CONTROL notification is sent to the owner of the container whenever the grid is resized.

```
param1
    USHORT id          /* Container control ID. */
    USHORT CN_GRIDRESIZED /* Notification code. */

param2
    SHORT sGridRows    /* New number of rows. */
    SHORT sGridCols    /* New number of columns. */
```

CN_GRIDRESIZED - Remarks

The grid is usually resized whenever the window is resized. This means there will be more or fewer grid squares than there were previously. After the resizing all grid squares are marked CM_AVAIL, regardless of their state before the resizing. The application must send another CM_SETGRIDINFO message to re-mark the squares as available or unavailable. Re-marking the grid squares is necessary only if the application wants to use the CMA_USER arrange pattern, otherwise the notification can be ignored.

CN_GRIDRESIZED - Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_GRIDRESIZED - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

CN_HELP

CN_HELP Field - id

id ([USHORT](#))
Container control ID.

CN_HELP Field - CN_HELP

CN_HELP ([USHORT](#))
Notification code.

CN_HELP Field - pRecord

pRecord ([PRECORDCORE](#))
Pointer to the record that has the selection cursor.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

CN_HELP Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_HELP - Parameters

id ([USHORT](#))
Container control ID.

CN_HELP ([USHORT](#))
Notification code.

pRecord ([PRECORDCORE](#))
Pointer to the record that has the selection cursor.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_HELP - Syntax

The container control sends a [WM_CONTROL \(in Container Controls\)](#) message with the CN_HELP notification code to its owner whenever the container receives a [WM_HELP](#) message.

```
param1
    USHORT      id          /* Container control ID. */
    USHORT      CN_HELP     /* Notification code. */

param2
    PRECORDCORE pRecord     /* Pointer to the record that has the selection cursor. */
```

CN_HELP - Remarks

This notification code is sent to the container's owner when help is requested for a container item.

CN_HELP - Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_HELP - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CN_INITDRAG

CN_INITDRAG Field - id

id ([USHORT](#))
Container control ID.

CN_INITDRAG Field - CN_INITDRAG

CN_INITDRAG ([USHORT](#))
Notification code.

CN_INITDRAG Field - pCnrDragInit

pCnrDragInit ([PCNRDRAGINIT](#))
Pointer to the [CNRDRAGINIT](#) structure.

CN_INITDRAG Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_INITDRAG - Parameters

id ([USHORT](#))

Container control ID.

CN_INITDRAG ([USHORT](#))

Notification code.

pCnrDragInit ([PCNRDRAGINIT](#))

Pointer to the [CNRDRAGINIT](#) structure.

ulReserved ([ULONG](#))

Reserved value, should be 0.

CN_INITDRAG - Syntax

The container control sends a [WM_CONTROL \(in Container Controls\)](#) message with the CN_INITDRAG notification code to its owner when the drag button is pressed and the pointer is moved while the pointer is over the container control.

```
param1
    USHORT      id          /* Container control ID. */
    USHORT      CN_INITDRAG /* Notification code. */

param2
    PCNRDRAGINIT pCnrDragInit /* Pointer to the CNRDRAGINIT structure. */
```

CN_INITDRAG - Remarks

This notification code is sent to the container's owner when the drag button is pressed and the pointer is moved while the pointer is over the container control.

CN_INITDRAG - Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_INITDRAG - Topics

Select an item:

CN_KILLFOCUS

CN_KILLFOCUS Field - id

id ([USHORT](#))
Container control ID.

CN_KILLFOCUS Field - CN_KILLFOCUS

CN_KILLFOCUS ([USHORT](#))
Notification code.

CN_KILLFOCUS Field - hwndCnr

hwndCnr ([HWND](#))
Container control handle.

CN_KILLFOCUS Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_KILLFOCUS - Parameters

id ([USHORT](#))
Container control ID.

CN_KILLFOCUS ([USHORT](#))
Notification code.

hwndCnr ([HWND](#))
Container control handle.

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_KILLFOCUS - Syntax

The container control sends a [WM_CONTROL \(in Container Controls\)](#) message with the CN_KILLFOCUS notification code to its owner whenever the container is losing the focus.

```
param1
    USHORT id          /* Container control ID. */
    USHORT CN_KILLFOCUS /* Notification code. */

param2
    HWND hwndCnr       /* Container control handle. */
```

CN_KILLFOCUS - Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_KILLFOCUS - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Default Processing](#)
- [Glossary](#)

CN_PICKUP

CN_PICKUP Field - id

id (USHORT)
Container control ID.

CN_PICKUP Field - CN_PICKUP

CN_PICKUP (USHORT)
Notification code.

CN_PICKUP Field - pCnrDragInit

pCnrDragInit (PCNRDRAGINIT)
Pointer to the CNRDRAGINIT structure containing direct-manipulation information initiated in a container.

The CNRDRAGINIT structure is the same as the one used for standard drag notifications.

CN_PICKUP Field - ulReserved

ulReserved (ULONG)
Reserved value, must be 0.

CN_PICKUP - Parameters

id (USHORT)
Container control ID.

CN_PICKUP (USHORT)
Notification code.

pCnrDragInit (PCNRDRAGINIT)
Pointer to the CNRDRAGINIT structure containing direct-manipulation information initiated in a container.

The CNRDRAGINIT structure is the same as the one used for standard drag notifications.

ulReserved (ULONG)
Reserved value, must be 0.

CN_PICKUP - Syntax

The container control sends a [WM_CONTROL \(in Container Controls\)](#) message with the CN_PICKUP notification code to its owner when a pickup and drop operation is initiated over a container.

```
param1
    USHORT      id          /* Container control ID. */
    USHORT      CN_PICKUP   /* Notification code. */

param2
    PCNRDRAGINIT pCnrDragInit /* Pointer to the CNRDRAGINIT structure containing direct-manipulation inform

returns
    ULONG      ulReserved /* Reserved value, must be 0. */
```

CN_PICKUP - Remarks

This notification code is sent to the owner of the container when a lazy drag operation is commenced over a container. (The container control receives a [WM_PICKUP](#) message.)

The CN_PICKUP message handler determines if the mouse is over an object or in white space of the client window.

If a pickup object is not selected, only that pickup object is added to the lazy drag set. If the pickup object is selected, all selected items in the container are added to the lazy drag set. The shell sets the CRA_PICKED attributes for all objects that are picked.

CN_PICKUP - Default Processing

The default message procedure sets *ulReserved* to 0.

CN_PICKUP - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CN_QUERYDELTA

CN_QUERYDELTA Field - id

id (USHORT)
Container control ID.

CN_QUERYDELTA Field - CN_QUERYDELTA

CN_QUERYDELTA (USHORT)
Notification code.

CN_QUERYDELTA Field - pNotifyDelta

pNotifyDelta (PNOTIFYDELTA)
Pointer to the NOTIFYDELTA structure.

CN_QUERYDELTA Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

CN_QUERYDELTA - Parameters

id (USHORT)
Container control ID.

CN_QUERYDELTA (USHORT)
Notification code.

pNotifyDelta (PNOTIFYDELTA)
Pointer to the NOTIFYDELTA structure.

ulReserved (ULONG)
Reserved value, should be 0.

CN_QUERYDELTA - Syntax

The container control sends a [WM_CONTROL \(in Container Controls\)](#) message with the CN_QUERYDELTA notification code to its owner to query for more data when a user scrolls to a preset delta value.

```
param1
    USHORT      id          /* Container control ID. */
    USHORT      CN_QUERYDELTA /* Notification code. */

param2
    PNOTIFYDELTA pNotifyDelta /* Pointer to the NOTIFYDELTA structure. */
```

CN_QUERYDELTA - Remarks

The delta value is specified by the *cDelta* field of the [CNRINFO](#) data structure and is set with the CMA_DELTA attribute of the [CM_SETCNRINFO](#) message. If the value of the *cDelta* field is greater than 0 and a user scrolls to the threshold record, the container control sends a CN_QUERYDELTA notification code to the application. The application can then insert more records into the container. It may be necessary for the application to remove some records before inserting records.

CN_QUERYDELTA - Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_QUERYDELTA - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

CN_REALLOCPSZ

CN_REALLOCPSZ Field - id

id ([USHORT](#))
Container control ID.

CN_REALLOCPSZ Field - CN_REALLOCPSZ

CN_REALLOCPSZ ([USHORT](#))
Notification code.

CN_REALLOCPSZ Field - pCnrEditData

pCnrEditData ([PCNREDITDATA](#))
Pointer to the [CNREDITDATA](#) structure.

CN_REALLOCPSZ Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	The application has sufficient memory for the new text string.
FALSE	The application has insufficient memory for the new text string or does not want the string to be copied.

CN_REALLOCPSZ - Parameters

id ([USHORT](#))
Container control ID.

CN_REALLOCPSZ ([USHORT](#))
Notification code.

pCnrEditData ([PCNREDITDATA](#))
Pointer to the [CNREDITDATA](#) structure.

rc ([BOOL](#))
Success indicator.

TRUE	The application has sufficient memory for the new text string.
FALSE	The application has insufficient memory for the new text string or does not want the string to be copied.

CN_REALLOCPSZ - Syntax

The container control sends a [WM_CONTROL \(in Container Controls\)](#) message with the CN_REALLOCPSZ notification code to its owner whenever container text is edited. It is sent before the [CN_ENDEDIT](#) notification code is sent.

```
param1
    USHORT      id          /* Container control ID. */
    USHORT      CN_REALLOCPSZ /* Notification code. */

param2
    PCNREDITDATA pCnrEditData /* Pointer to the CNREDITDATA structure. */
```

CN_REALLOCPSZ - Remarks

The CN_REALLOCPSZ notification code is sent after direct editing of container text is complete. It notifies the application that the container is about to copy the changed text to the application's text string. This allows the application to ensure that the correct amount of memory is allocated to accommodate the change.

If TRUE is returned by the application, the container control copies the new text to the application's text string. However, if the application returns FALSE, changed text is disregarded.

Warning: Once your application receives the CN_REALLOCPSZ notification code, it must not send any messages to the container until it receives the [CN_ENDEDIT](#) notification code, which indicates that direct editing of container text has ended. If any messages are sent to the container before your application receives the [CN_ENDEDIT](#) notification code, the results of direct editing are unpredictable.

CN_REALLOCPSZ - Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return FALSE.

CN_REALLOCPSZ - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CN_SCROLL

CN_SCROLL Field - id

id ([USHORT](#))
Container control ID.

CN_SCROLL Field - CN_SCROLL

CN_SCROLL ([USHORT](#))
Notification code.

CN_SCROLL Field - pNotifyScroll

pNotifyScroll ([PNOTIFYSCROLL](#))
Pointer to the [NOTIFYSCROLL](#) structure.

CN_SCROLL Return Value - rc

rc ([ULONG](#))
Reserved value, should be 0.

CN_SCROLL - Parameters

id ([USHORT](#))
Container control ID.

CN_SCROLL ([USHORT](#))
Notification code.

pNotifyScroll ([PNOTIFYSCROLL](#))
Pointer to the [NOTIFYSCROLL](#) structure.

rc ([ULONG](#))
Reserved value, should be 0.

CN_SCROLL - Syntax

The container control sends a [WM_CONTROL \(in Container Controls\)](#) message with the CN_SCROLL notification code to its owner whenever the container window scrolls.

```
param1
    USHORT      id          /* Container control ID. */
    USHORT      CN_SCROLL   /* Notification code. */

param2
    NOTIFY_SCROLL pNotifyScroll /* Pointer to the NOTIFY_SCROLL structure. */
```

CN_SCROLL - Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_SCROLL - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Default Processing](#)
 - [Glossary](#)

CN_SETFOCUS

CN_SETFOCUS Field - id

id ([USHORT](#))
Container control ID.

CN_SETFOCUS Field - CN_SETFOCUS

CN_SETFOCUS ([USHORT](#))
Notification code.

CN_SETFOCUS Field - hwndCnr

hwndCnr ([HWND](#))
Container control handle.

CN_SETFOCUS Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_SETFOCUS - Parameters

id ([USHORT](#))
Container control ID.

CN_SETFOCUS ([USHORT](#))
Notification code.

hwndCnr ([HWND](#))
Container control handle.

ulReserved ([ULONG](#))
Reserved value, should be 0.

CN_SETFOCUS - Syntax

The container control sends a [WM_CONTROL \(in Container Controls\)](#) message with the CN_SETFOCUS notification code to its owner whenever the container receives the focus.

```
param1
    USHORT id          /* Container control ID. */
    USHORT CN_SETFOCUS /* Notification code. */

param2
    HWND hwndCnr       /* Container control handle. */
```

CN_SETFOCUS - Default Processing

The default window procedure does not expect to receive this notification code and therefore takes no action on it other than to return 0.

CN_SETFOCUS - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Default Processing](#)
 - [Glossary](#)

Container Control Window Messages

This section describes the container control window procedure actions on receiving the following messages.

CM_ALLOCDETAILFIELDINFO

CM_ALLOCDETAILFIELDINFO Field - nFieldInfo

nFieldInfo ([USHORT](#))
Number of [FIELDINFO](#) structures to be allocated.

The value of this parameter must be greater than 0.

CM_ALLOCDETAILFIELDINFO Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

CM_ALLOCDETAILFIELDINFO Return Value - pFieldInfo

pFieldInfo (PFIELDINFO)

Pointer or error.

0

Reserved value, 0. The [WinGetLastError](#) function may return the following errors:

- PMERR_INSUFFICIENT_MEMORY
- PMERR_INVALID_PARAMETERS.

Other

If the *nFieldInfo* parameter has a value of 1, a pointer to a [FIELDINFO](#) data structure is returned.

A pointer to the first [FIELDINFO](#) structure in a linked list of [FIELDINFO](#) structures is returned if the *nFieldInfo* parameter has a value greater than 1. The pointer to the next [FIELDINFO](#) structure is set in each *pNextFieldInfo* field of the [FIELDINFO](#) data structure. The last pointer is set to NULL.

CM_ALLOCDETAILFIELDINFO - Parameters

nFieldInfo (USHORT)

Number of [FIELDINFO](#) structures to be allocated.

The value of this parameter must be greater than 0.

ulReserved (ULONG)

Reserved value, should be 0.

pFieldInfo (PFIELDINFO)

Pointer or error.

0

Reserved value, 0. The [WinGetLastError](#) function may return the following errors:

- PMERR_INSUFFICIENT_MEMORY
- PMERR_INVALID_PARAMETERS.

Other

If the *nFieldInfo* parameter has a value of 1, a pointer to a [FIELDINFO](#) data structure is returned.

A pointer to the first [FIELDINFO](#) structure in a linked list of [FIELDINFO](#) structures is returned if the *nFieldInfo* parameter has a value greater than 1. The pointer to the next [FIELDINFO](#) structure is set in each *pNextFieldInfo* field of the [FIELDINFO](#) data structure. The last pointer is set to NULL.

CM_ALLOCDETAILFIELDINFO - Syntax

This message allocates memory for one or more [FIELDINFO](#) structures.

```
param1
    USHORT    nFieldInfo /* Number of FIELDINFO structures to be allocated. */
```


param2
 ULONG **ulReserved** /* Reserved value, should be 0. */

CM_ALLOCDETAILFIELDINFO - Remarks

The container control requires that the application use the CM_ALLOCDETAILFIELDINFO message to allocate memory for any **FIELDINFO** structures that are used.

CM_ALLOCDETAILFIELDINFO - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return NULL.

CM_ALLOCDETAILFIELDINFO - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

CM_ALLOCRECORD

CM_ALLOCRECORD Field - cbRecordData

cbRecordData (**ULONG**)
Bytes of additional memory.

The number of bytes of additional memory that you want to reserve for your application's private use. This parameter must have a value between 0 and 64,000. If the value is 0, no additional memory is allocated, but a **RECORDCORE** data structure is allocated.

CM_ALLOCRECORD Field - nRecords

nRecords ([USHORT](#))

Number of records.

The number of container records to be allocated. This parameter must have a value greater than 0.

CM_ALLOCRECORD Return Value - pRecord

pRecord ([PRECORDCORE](#))

Returns a pointer or an error.

NULL

Allocation failed. The [WinGetLastError](#) function may return the following errors:

- [PMERR_INSUFFICIENT_MEMORY](#)
- [PMERR_INVALID_PARAMETERS](#).

Other

If the *nRecords* parameter has a value of 1, a pointer to a [RECORDCORE](#) structure is returned.

If the *nRecords* parameter has a value greater than 1, a pointer to the first [RECORDCORE](#) structure in the linked list of records is returned. The pointer to the next container record is set in the *preccNextRecord* field in each [RECORDCORE](#) data structure. The last pointer is set to NULL.

CM_ALLOCRECORD - Parameters

cbRecordData ([ULONG](#))

Bytes of additional memory.

The number of bytes of additional memory that you want to reserve for your application's private use. This parameter must have a value between 0 and 64,000. If the value is 0, no additional memory is allocated, but a [RECORDCORE](#) data structure is allocated.

nRecords ([USHORT](#))

Number of records.

The number of container records to be allocated. This parameter must have a value greater than 0.

pRecord ([PRECORDCORE](#))

Returns a pointer or an error.

NULL

Allocation failed. The [WinGetLastError](#) function may return the following errors:

- [PMERR_INSUFFICIENT_MEMORY](#)
- [PMERR_INVALID_PARAMETERS](#).

Other

If the *nRecords* parameter has a value of 1, a pointer to a [RECORDCORE](#) structure is returned.

If the *nRecords* parameter has a value greater than 1, a pointer to the first [RECORDCORE](#) structure in the linked list of records is returned. The pointer to the next container record is set in the *preccNextRecord* field in each [RECORDCORE](#) data structure. The last pointer is set to NULL.

CM_ALLOCRECORD - Syntax

This message allocates memory for one or more [RECORDCORE](#) structures.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

```
param1
    ULONG          cbRecordData /* Bytes of additional memory. */

param2
    USHORT         nRecords     /* Number of records. */
```

CM_ALLOCRECORD - Remarks

The container control requires that the application use the CM_ALLOCRECORD message to allocate memory for container records.

When a record is allocated, the *cb* field of the record will be initialized with the size of the record structure type currently in use, either [RECORDCORE](#) or [MINIRECORDCORE](#). If the CCS_MINIRECORDCORE style bit is not specified, the record is allocated according to the size of the [RECORDCORE](#) data structure. However, if the CCS_MINIRECORDCORE style bit is specified, the record is allocated according to the size of the [MINIRECORDCORE](#) data structure. This size should not be modified by the application.

CM_ALLOCRECORD - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return NULL.

CM_ALLOCRECORD - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

CM_ARRANGE

CM_ARRANGE Field - ulArrangeType

ulArrangeType (ULONG)

Type of arrangement for icons or bitmaps.

Specify one of the following arrangement types:

CMA_ARRANGESTANDARD

Standard arrangement.

Icons are arranged at the top of the window, starting at the upper left-hand corner. This value is set to zero to provide compatibility with existing applications.

CMA_ARRANGEGRID

Grid arrangement.

Icons are arranged in a grid of equal-size squares in the container window. The grid pattern can be defined as the perimeter, left, right, top, or bottom of the window. It also can be user-defined.

CMA_ARRANGESELECTED

Selected arrangement.

Currently selected icons are arranged horizontally or vertically in the container window.

CM_ARRANGE Field - ulArrangeFlags

ulArrangeFlags (ULONG)

Flags for arrangement type specified in *ulType*.

CMA_ARRANGESTANDARD

If the arrangement type is CMA_ARRANGESTANDARD, this value is NULL.

CMA_ARRANGEGRID

If the arrangement type is CMA_ARRANGEGRID, specify one of the following arrangement patterns:

CMA_PERIMETER

Icons are arranged along the top, left and right side of the window, leaving the middle unoccupied.

CMA_LEFT

Icons are arranged starting at the upper left corner of the window and proceeding top-to-bottom, left-to-right.

CMA_RIGHT

Icons are arranged starting at the upper right corner of the window, and proceeding top-to-bottom, right-to-left.

CMA_TOP

Icons are arranged at the top of the window. The difference between this and the standard arrangement is that positioning is according to grid square size, not tile size. (An icon's tile size is the rectangle that encompasses both icon and text.)

CMA_BOTTOM

Icons are arranged, starting at the lower left corner and proceeding left-to-right, bottom-to-top.

CMA_USER

The application provides the user with a means of specifying which squares are to be occupied and which are not. The CM_SETGRIDINFO message marks each grid square as available or unavailable for an icon. The current grid information can be obtained at any time by using the CM_QUERYGRIDINFO message.

CMA_ARRANGESELECTED

If the arrangement type is CMA_ARRANGESELECTED, specify one of the following orientations:

CMA_HORIZONTAL	Orient selected icons horizontally.
CMA_VERTICAL	Orient selected icons vertically.

When the container receives a message indicating a selected arrangement is desired, it goes into "Selected Arrange" mode. The mouse is captured and the pointer changes to indicate the orientation of the arrangement selection in progress (horizontal or vertical). When the user presses mouse button 1 over an area in the container window, the icons are immediately positioned at that point, and the container ends the selected range mode and releases the mouse capture. If the current view is CV_ICON, the selected icons are arranged starting at the precise point where the user clicked the mouse. If the current view is CV_GRID, the selected icons are arranged starting at the grid square closest to the point where the user clicked the mouse.

CM_ARRANGE Return Value - rc

rc (BOOL)

Success indicator.

TRUE

Icon/text or bit-map/text pairs were successfully arranged.

FALSE

An error occurred.

CM_ARRANGE - Parameters

ulArrangeType (ULONG)

Type of arrangement for icons or bitmaps.

Specify one of the following arrangement types:

CMA_ARRANGESTANDARD

Standard arrangement.

Icons are arranged at the top of the window, starting at the upper left-hand corner. This value is set to zero to provide compatibility with existing applications.

CMA_ARRANGEGRID

Grid arrangement.

Icons are arranged in a grid of equal-size squares in the container window. The grid pattern can be defined as the perimeter, left, right, top, or bottom of the window. It also can be user-defined.

CMA_ARRANGESELECTED

Selected arrangement.

Currently selected icons are arranged horizontally or vertically in the container window.

ulArrangeFlags (ULONG)

Flags for arrangement type specified in *ultype*.

CMA_ARRANGESTANDARD

If the arrangement type is CMA_ARRANGESTANDARD, this value is NULL.

CMA_ARRANGEGRID

If the arrangement type is CMA_ARRANGEGRID, specify one of the following arrangement patterns:

CMA_PERIMETER

Icons are arranged along the top, left and right side of the window, leaving the middle unoccupied.

CMA_LEFT

Icons are arranged starting at the upper left corner

	of the window and proceeding top-to-bottom, left-to-right.
CMA_RIGHT	Icons are arranged starting at the upper right corner of the window, and proceeding top-to-bottom, right-to-left.
CMA_TOP	Icons are arranged at the top of the window. The difference between this and the standard arrangement is that positioning is according to grid square size, not tile size. (An icon's tile size is the rectangle that encompasses both icon and text.)
CMA_BOTTOM	Icons are arranged, starting at the lower left corner and proceeding left-to-right, bottom-to-top.
CMA_USER	The application provides the user with a means of specifying which squares are to be occupied and which are not. The CM_SETGRIDINFO message marks each grid square as available or unavailable for an icon. The current grid information can be obtained at any time by using the CM_QUERYGRIDINFO message.
CMA_ARRANGESELECTED	If the arrangement type is CMA_ARRANGESELECTED, specify one of the following orientations:
CMA_HORIZONTAL	Orient selected icons horizontally.
CMA_VERTICAL	Orient selected icons vertically.
<p>When the container receives a message indicating a selected arrangement is desired, it goes into "Selected Arrange" mode. The mouse is captured and the pointer changes to indicate the orientation of the arrangement selection in progress (horizontal or vertical). When the user presses mouse button 1 over an area in the container window, the icons are immediately positioned at that point, and the container ends the selected range mode and releases the mouse capture. If the current view is CV_ICON, the selected icons are arranged starting at the precise point where the user clicked the mouse. If the current view is CV_GRID, the selected icons are arranged starting at the grid square closest to the point where the user clicked the mouse.</p>	
rc (BOOL)	Success indicator.
TRUE	Icon/text or bit-map/text pairs were successfully arranged.
FALSE	An error occurred.

CM_ARRANGE - Syntax

This message arranges the container records in the icon view of the container control.

```
param1
    ULONG ulArrangeType /* Type of arrangement for icons or bitmaps. */

param2
    ULONG ulArrangeFlags /* Flags for arrangement type specified in ultype. */
```

CM_ARRANGE - Remarks

A vertical scroll bar is enabled, if necessary.

Before the relocation of the container items, the origin of the client area rectangle is reset to coincide with the origin of the container's workspace. Arranging the container items does not affect the record attributes.

If the CCS_AUTOPOSITION style bit is set, you do not need to send the CM_ARRANGE message, since this style bit causes the container control to arrange the container items for the application.

If the current view is not the icon view, no visible change occurs until the current view is switched to the icon view. For example, if the name view is the current view and the CM_ARRANGE message is sent, the display does not change.

The container updates the *pillcon* field of the [RECORDCORE](#) structure with the new coordinates.

CM_ARRANGE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_ARRANGE - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

CM_CLOSEEDIT

CM_CLOSEEDIT Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

CM_CLOSEEDIT Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

CM_CLOSEEDIT Return Value - rc

rc (BOOL)	Success indicator.
TRUE	The direct editing of container item text was successfully ended.
FALSE	The direct editing of container item text was not successfully ended. The WinGetLastError function may return the following error: PMERR_INSUFFICIENT_MEMORY.

CM_CLOSEEDIT - Parameters

ulReserved (ULONG)	Reserved value, should be 0.
ulReserved (ULONG)	Reserved value, should be 0.
rc (BOOL)	Success indicator.
TRUE	The direct editing of container item text was successfully ended.
FALSE	The direct editing of container item text was not successfully ended. The WinGetLastError function may return the following error: PMERR_INSUFFICIENT_MEMORY.

CM_CLOSEEDIT - Syntax

This message closes the window that contains the multiple-line entry (MLE) field used to edit container text directly.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

CM_CLOSEEDIT - Remarks

The application sends this message to the container control to end the direct editing of container text. The application can assign this message to a key or key combination, a menu choice, or both so that the user can end the direct editing of container text from the keyboard.

When the container control receives this message, it sends the [CN_REALLOCPSZ](#) and [CN_ENDEDIT](#) notification codes to the application.

CM_CLOSEEDIT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_CLOSEEDIT - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CM_COLLAPSETREE

CM_COLLAPSETREE Field - pRecord

pRecord ([PRECORDCORE](#))

Pointer to the [RECORDCORE](#) structure that is to be collapsed.

If this is NULL, all expanded parent items are collapsed.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

CM_COLLAPSETREE Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

CM_COLLAPSETREE Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE The item was successfully collapsed.

FALSE An error occurred. The [WinGetLastError](#) function may return the following error:
 PMERR_INVALID_PARAMETERS.

CM_COLLAPSETREE - Parameters

pRecord ([PRECORDCORE](#))
Pointer to the [RECORDCORE](#) structure that is to be collapsed.

If this is NULL, all expanded parent items are collapsed.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Success indicator.

TRUE The item was successfully collapsed.

FALSE An error occurred. The [WinGetLastError](#) function may return the following error:
 PMERR_INVALID_PARAMETERS.

CM_COLLAPSETREE - Syntax

This message causes one parent item in the tree view to be collapsed.

```
param1
    PRECORDCORE  pRecord      /* Pointer to the RECORDCORE structure that is to be collapsed. */

param2
    ULONG         ulReserved  /* Reserved value, should be 0. */
```

CM_COLLAPSETREE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_COLLAPSETREE - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Default Processing
 - Glossary

CM_ERASERECORD

CM_ERASERECORD Field - pRecord

pRecord ([PRECORDCORE](#))
Pointer to the container record that is to be erased from the current view.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

CM_ERASERECORD Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

CM_ERASERECORD Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	The record was successfully erased.
FALSE	

The record was not erased. The [WinGetLastError](#) function may return the following errors:

- [PMERR_INVALID_PARAMETERS](#)
- [PMERR_INSUFFICIENT_MEMORY](#).

CM_ERASERECORD - Parameters

pRecord ([PRECORDCORE](#))

Pointer to the container record that is to be erased from the current view.

Note: If the [CCS_MINIRECORDCORE](#) style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and [PMINIRECORDCORE](#) should be used instead of [PRECORDCORE](#) in all applicable data structures and messages.

ulReserved ([ULONG](#))

Reserved value, should be 0.

rc ([BOOL](#))

Success indicator.

TRUE

The record was successfully erased.

FALSE

The record was not erased. The [WinGetLastError](#) function may return the following errors:

- [PMERR_INVALID_PARAMETERS](#)
- [PMERR_INSUFFICIENT_MEMORY](#).

CM_ERASERECORD - Syntax

This message erases the source record from the current view when a move occurs as a result of direct manipulation.

```
param1
    PRECORDCORE  pRecord      /* Pointer to the container record that is to be erased from the current view. */
param2
    ULONG         ulReserved  /* Reserved value, should be 0. */
```

CM_ERASERECORD - Remarks

The container record is not removed and memory is not freed; only the visual appearance is changed. The visibility flag associated with the container record is not changed.

CM_ERASERECORD - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return [FALSE](#).

CM_ERASERECORD - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

CM_EXPANDTREE

CM_EXPANDTREE Field - pRecord

pRecord ([PRECORDCORE](#))
Pointer to the [RECORDCORE](#) structure that is to be expanded.

If this is NULL, all collapsed parent items are expanded.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

CM_EXPANDTREE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

CM_EXPANDTREE Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	The item was successfully expanded.
FALSE	An error occurred. The WinGetLastError function may return the following error:

PMERR_INVALID_PARAMETERS.

CM_EXPANDTREE - Parameters

pRecord (PRECORDCORE)

Pointer to the [RECORDCORE](#) structure that is to be expanded.

If this is NULL, all collapsed parent items are expanded.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

ulReserved (ULONG)

Reserved value, should be 0.

rc (BOOL)

Success indicator.

TRUE

The item was successfully expanded.

FALSE

An error occurred. The [WinGetLastError](#) function may return the following error:

PMERR_INVALID_PARAMETERS.

CM_EXPANDTREE - Syntax

This message causes one parent item in the tree view to be expanded.

```
param1
    PRECORDCORE pRecord    /* Pointer to the RECORDCORE structure that is to be expanded. */
param2
    ULONG          ulReserved /* Reserved value, should be 0. */
```

CM_EXPANDTREE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_EXPANDTREE - Topics

Select an item:

CM_FILTER

CM_FILTER Field - pfnFilter

pfnFilter ([PFN](#))
Pointer to an application-supplied filter function.

CM_FILTER Field - pStorage

pStorage ([PVOID](#))
Application use.

Available for application use.

CM_FILTER Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	A subset was successfully created.
FALSE	An error occurred. The WinGetLastError function may return the following errors: <ul style="list-style-type: none">PMERR_NO_FILTERED_ITEMSPMERR_INSUFFICIENT_MEMORY.

CM_FILTER - Parameters

pfnFilter (PFN)

Pointer to an application-supplied filter function.

pStorage (PVOID)

Application use.

Available for application use.

rc (BOOL)

Success indicator.

TRUE

A subset was successfully created.

FALSE

An error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_NO_FILTERED_ITEMS
- PMERR_INSUFFICIENT_MEMORY.

CM_FILTER - Syntax

This message filters the contents of a container so that a subset of the container items is viewable.

```
param1
    PFN    pfnFilter /* Pointer to an application-supplied filter function. */

param2
    PVOID  pStorage /* Application use. */
```

CM_FILTER - Remarks

Filtering is enabled by setting the CRA_FILTERED attribute of container records that are to be excluded from the viewable subset.

The *pfnFilter* parameter points to an application-provided function that determines whether a record is to be included in the viewable subset. The *pfnFilter* parameter must be declared as:

```
BOOL PFN pfnFilter (
    PRECORDCORE p,
    PVOID pStorage);
```

where **p** points to a [RECORDCORE](#) structure that describes the container record to be tested. The *pfnFilter* parameter returns TRUE if the record is to be included in the viewable subset, or FALSE if it is to be excluded. The container sets the CRA_FILTERED attribute for the record based on the return from the *pfnFilter* parameter.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

If the CRA_FILTERED attribute is set for the record, the record is not visible. If the CCS_AUTOPOSITION style bit is set and the container is showing the icon view, the container records are arranged when a record is filtered out.

The CM_FILTER message supports only one level of filtering.

It is the application's responsibility to provide a National Language Support-enabled (NLS-enabled) function for the *pfnFilter* parameter.

If the *pfnFilter* parameter value is NULL, a container is returned to an unfiltered state. If functions such as inserting a record into a container, arranging the records, or sorting the records are performed on a container whose records have been filtered, the effect of these functions remains if the container records are later unfiltered.

All messages act on the entire container. For example, a record that is filtered and is removed from the container will be removed from the container entirely; it is not present in the container when the container records are unfiltered.

CM_FILTER - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_FILTER - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

CM_FREEDETAILELDFIELDINFO

CM_FREEDETAILELDFIELDINFO Field - pFieldInfoArray

pFieldInfoArray ([PVOID](#))
Pointer to an array of pointers to [FIELDINFO](#) structures that are to be freed.

CM_FREEDETAILELDFIELDINFO Field - cNumFieldInfo

cNumFieldInfo ([USHORT](#))
Number of structures.

Number of [FIELDINFO](#) structures to be freed.

CM_FREEDETAILELDFIELDINFO Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE
Memory associated with a specified [FIELDINFO](#) structure or structures in the container was freed.

FALSE
Associated memory was not freed. The [WinGetLastError](#) function may return the following errors:

- [PMERR_INVALID_PARAMETERS](#)
- [PMERR_MEMORY_DEALLOCATION_ERR](#)
- [PMERR_FI_CURRENTLY_INSERTED](#).

CM_FREEDetailFIELDINFO - Parameters

pFieldInfoArray ([PVOID](#))
Pointer to an array of pointers to [FIELDINFO](#) structures that are to be freed.

cNumFieldInfo ([USHORT](#))
Number of structures.

Number of [FIELDINFO](#) structures to be freed.

rc ([BOOL](#))
Success indicator.

TRUE
Memory associated with a specified [FIELDINFO](#) structure or structures in the container was freed.

FALSE
Associated memory was not freed. The [WinGetLastError](#) function may return the following errors:

- [PMERR_INVALID_PARAMETERS](#)
- [PMERR_MEMORY_DEALLOCATION_ERR](#)
- [PMERR_FI_CURRENTLY_INSERTED](#).

CM_FREEDetailFIELDINFO - Syntax

This message frees the memory associated with one or more [FIELDINFO](#) structures.

```
param1
    PVOID    pFieldInfoArray /* Pointer to an array of pointers to FIELDINFO structures that are to be freed. */

param2
    USHORT   cNumFieldInfo   /* Number of structures. */
```

CM_FREEDetailFIELDINFO - Remarks

It is the application's responsibility to free all application-allocated memory associated with the structures, such as user data.

If a specified [FIELDINFO](#) structure is currently inserted into the container, the structure is not freed and the PMERR_FI_CURRENTLY_INSERTED error is set. [FIELDINFO](#) structures must be removed with the [CM_REMOVEDTAILFIELDINFO](#) message before the CM_FREEDetailFIELDINFO message is used.

If the number of pointers to [FIELDINFO](#) structures in the array exceeds the count of structures to be freed, only the number of structures in the *cNumFieldInfo* parameter is freed. If either the *pFieldInfoArray* or the *cNumFieldInfo* parameter is invalid, the PMERR_INVALID_PARAMETERS error is set and no [FIELDINFO](#) structures are freed.

If the PMERR_MEMORY_DEALLOCATION_ERR error occurs, any further processing is unreliable.

CM_FREEDetailFIELDINFO - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_FREEDetailFIELDINFO - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

CM_FREERECORD

CM_FREERECORD Field - pRecordArray

pRecordArray ([PVOID](#))
Pointer to an array of pointers to [RECORDCORE](#) structures that are to be freed.

CM_FREERECORD Field - cNumRecord

cNumRecord ([USHORT](#))
Number of records.

Number of container records to be freed.

CM_FREERECORD Return Value - rc

rc ([BOOL](#))

Success indicator.

TRUE

Memory associated with a record or records in the container was freed.

FALSE

Associated memory was not freed. The [WinGetLastError](#) function may return the following errors:

- [PMERR_INVALID_PARAMETERS](#)
- [PMERR_MEMORY_DEALLOCATION_ERR](#)
- [PMERR_RECORD_CURRENTLY_INSERTED](#).

CM_FREERECORD - Parameters

pRecordArray ([PVOID](#))

Pointer to an array of pointers to [RECORDCORE](#) structures that are to be freed.

cNumRecord ([USHORT](#))

Number of records.

Number of container records to be freed.

rc ([BOOL](#))

Success indicator.

TRUE

Memory associated with a record or records in the container was freed.

FALSE

Associated memory was not freed. The [WinGetLastError](#) function may return the following errors:

- [PMERR_INVALID_PARAMETERS](#)
- [PMERR_MEMORY_DEALLOCATION_ERR](#)
- [PMERR_RECORD_CURRENTLY_INSERTED](#).

CM_FREERECORD - Syntax

This message frees the memory associated with one or more [RECORDCORE](#) structures.

Note: If the [CCS_MINIRECORDCORE](#) style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and [PMINIRECORDCORE](#) should be used instead of [PRECORDCORE](#) in all applicable data structures and messages.

```
param1
    VOID    pRecordArray /* Pointer to an array of pointers to RECORDCORE structures that are to be freed. */
param2
    SHORT   cNumRecord   /* Number of records. */
```

CM_FREERECORD - Remarks

It is the application's responsibility to free all application-allocated memory associated with the container records, such as text strings.

If a specified record is currently inserted into the container, the record is not freed and the **PMERR_RECORD_CURRENTLY_INSERTED** error is set. Container records must be removed with the **CM_REMOVERECORD** message before the **CM_FREERECORD** message is used.

If the number of pointers to container records in the array exceeds the count of records to be freed, only the number of records in the *cNumRecord* parameter is freed. If either the *pRecordArray* or the *cNumRecord* parameter is invalid, the **PMERR_INVALID_PARAMETERS** error is set and no container records are freed.

If the **PMERR_MEMORY_DEALLOCATION_ERR** error occurs, any further processing is unreliable.

CM_FREERECORD - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return **FALSE**.

CM_FREERECORD - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CM_HORIZSCROLLSPLITWINDOW

CM_HORIZSCROLLSPLITWINDOW Field - usWindow

usWindow (**USHORT**)
Window indicator.

CMA_LEFT	The left split window is scrolled.
CMA_RIGHT	The right split window is scrolled.

CM_HORIZSCROLLSPLITWINDOW Field - IScrollInc

IScrollInc ([LONG](#))
Amount to scroll.

Amount (in pixels) by which to scroll the window.

CM_HORIZSCROLLSPLITWINDOW Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE
Successful completion

FALSE
An error occurred. The [WinGetLastError](#) function may return the following error:
PMERR_INVALID_PARAMETERS.

CM_HORIZSCROLLSPLITWINDOW - Parameters

usWindow ([USHORT](#))
Window indicator.

CMA_LEFT
The left split window is scrolled.

CMA_RIGHT
The right split window is scrolled.

IScrollInc ([LONG](#))
Amount to scroll.

Amount (in pixels) by which to scroll the window.

rc ([BOOL](#))
Success indicator.

TRUE
Successful completion

FALSE
An error occurred. The [WinGetLastError](#) function may return the following error:
PMERR_INVALID_PARAMETERS.

CM_HORIZSCROLLSPLITWINDOW - Syntax

This message scrolls a split window in the split details view.

```
param1
    USHORT    usWindow    /* Window indicator. */

param2
    LONG      lScrollInc  /* Amount to scroll. */
```

CM_HORIZSCROLLSPLITWINDOW - Remarks

The *lScrollInc* parameter indicates a change in position. If the *lScrollInc* parameter value is greater than 0, the window specified in the *usWindow* parameter is scrolled to the right by the number of pixels specified in the *lScrollInc* parameter. If the value of the *lScrollInc* parameter is less than 0, the window specified in the *usWindow* parameter is scrolled to the left by the number of pixels specified in the *lScrollInc* parameter. This message is used to scroll either the left or right split window by an absolute amount.

The columns that are to appear in each split window are determined at the time the split window is created. Thereafter, columns in the left split window cannot be seen in the right split window, and vice versa.

CM_HORIZSCROLLSPLITWINDOW - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_HORIZSCROLLSPLITWINDOW - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CM_INSERTDETAILFIELDINFO

CM_INSERTDETAILFIELDINFO Field - pFieldInfo

pFieldInfo ([PFIELDINFO](#))
Pointer to the [FIELDINFO](#) structure or structures to insert.

CM_INSERTDETAILFIELDINFO Field - pFieldInfoInsert

pFieldInfoInsert ([PFIELDINFOINSERT](#))
Pointer to the [FIELDINFOINSERT](#) data structure.

CM_INSERTDETAILFIELDINFO Return Value - cFields

cFields ([USHORT](#))
Number of structures.

0

The [FIELDINFO](#) structure or structures were not inserted. The [WinGetLastError](#) function may return the following errors:

- [PMERR_INVALID_PARAMETERS](#)
- [PMERR_INSUFFICIENT_MEMORY](#)
- [PMERR_FI_CURRENTLY_INSERTED](#).

Other

The number of [FIELDINFO](#) structures in the container.

CM_INSERTDETAILFIELDINFO - Parameters

pFieldInfo ([PFIELDINFO](#))
Pointer to the [FIELDINFO](#) structure or structures to insert.

pFieldInfoInsert ([PFIELDINFOINSERT](#))
Pointer to the [FIELDINFOINSERT](#) data structure.

cFields ([USHORT](#))
Number of structures.

0

The [FIELDINFO](#) structure or structures were not inserted. The [WinGetLastError](#) function may return the following errors:

- [PMERR_INVALID_PARAMETERS](#)
- [PMERR_INSUFFICIENT_MEMORY](#)
- [PMERR_FI_CURRENTLY_INSERTED](#).

Other

The number of [FIELDINFO](#) structures in the container.

CM_INSERTDETAILFIELDINFO - Syntax

This message inserts one or more [FIELDINFO](#) structures into a container control.

```
param1
    PFIELDINFO      pFieldInfo      /* Pointer to the FIELDINFO structure or structures to insert. */

param2
    PFIELDINFOINSERT pFieldInfoInsert /* Pointer to the FIELDINFOINSERT data structure. */
```

CM_INSERTDETAILFIELDINFO - Remarks

The *pFieldInfoInsert* parameter is used to insert [FIELDINFO](#) structures into the container. The *pFieldInfoOrder* field of the [FIELDINFOINSERT](#) data structure is used to place [FIELDINFO](#) structures into the container in order, relative to the other structures. Specifying the CMA_FIRST attribute places the [FIELDINFO](#) structure at the front of the list of structures. If the CMA_END attribute is specified, the [FIELDINFO](#) structure is placed at the end of the list of structures. Otherwise, if the value of the *pFieldInfoOrder* field is a pointer to a [FIELDINFO](#) structure, the structure being inserted is placed after this structure.

If the value of the *cFieldInfoInsert* field of the [FIELDINFOINSERT](#) data structure is greater than 1, a linked list of [FIELDINFO](#) structures is inserted in the order specified by the *pFieldInfoOrder* field. Here, the *pFieldInfo* parameter points to the first of a linked list of [FIELDINFO](#) structures. This list of structures is linked together as they were when the [FIELDINFO](#) structures were allocated.

If one [FIELDINFO](#) structure is to be inserted, the *cFieldInfoInsert* field has a value of 1 and the *pFieldInfo* parameter points to the [FIELDINFO](#) structure to be inserted.

After the [FIELDINFO](#) structures have been inserted, if the *flinvalidateFieldInfo* field of the [FIELDINFOINSERT](#) data structure is FALSE, the [CM_INVALIDATEDDETAILFIELDINFO](#) message must be sent to update the display with the inserted structures.

If the CCS_VERIFYPOINTERS style bit is set and the *pFieldInfo* parameter contains a pointer to a [FIELDINFO](#) structure that is currently inserted, the PMERR_FI_CURRENTLY_INSERTED error is set and no [FIELDINFO](#) structures are inserted.

CM_INSERTDETAILFIELDINFO - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

CM_INSERTDETAILFIELDINFO - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

CM_INSERTRECORD

CM_INSERTRECORD Field - pRecord

pRecord ([PRECORDCORE](#))

Pointer to the [RECORDCORE](#) structure or structures to insert.

If you set the *pRecord* field to a specific PRECORDCORE (rather than to CMA_FIRST or CMA_END), you must set the parent to NULL. If you do not do this and are inserting at the top level in tree view, the application will trap in PMCTLS.DLL.

CM_INSERTRECORD Field - pRecordInsert

pRecordInsert ([PRECORDINSERT](#))

Pointer to the [RECORDINSERT](#) data structure.

CM_INSERTRECORD Return Value - cRecords

cRecords ([ULONG](#))

Number of structures.

0

The [RECORDCORE](#) structure was not inserted. The [WinGetLastError](#) function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_INSUFFICIENT_MEMORY
- PMERR_RECORD_CURRENTLY_INSERTED.

Other

The number of [RECORDCORE](#) structures in the container.

CM_INSERTRECORD - Parameters

pRecord ([PRECORDCORE](#))

Pointer to the [RECORDCORE](#) structure or structures to insert.

If you set the *pRecord* field to a specific PRECORDCORE (rather than to CMA_FIRST or CMA_END), you must set the parent to NULL. If you do not do this and are inserting at the top level in tree view, the application will trap in PMCTLS.DLL.

pRecordInsert ([PRECORDINSERT](#))

Pointer to the [RECORDINSERT](#) data structure.

cRecords (ULONG)

Number of structures.

0

The [RECORDCORE](#) structure was not inserted. The [WinGetLastError](#) function may return the following errors:

- [PMERR_INVALID_PARAMETERS](#)
- [PMERR_INSUFFICIENT_MEMORY](#)
- [PMERR_RECORD_CURRENTLY_INSERTED](#).

Other

The number of [RECORDCORE](#) structures in the container.

CM_INSERTRECORD - Syntax

This message inserts one or more [RECORDCORE](#) structures into a container control.

Note: If the [CCS_MINIRECORDCORE](#) style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and [PMINIRECORDCORE](#) should be used instead of [PRECORDCORE](#) in all applicable data structures and messages.

```
param1
    PRECORDCORE    pRecord        /* Pointer to the RECORDCORE structure or structures to insert. */

param2
    PRECORDINSERT  pRecordInsert /* Pointer to the RECORDINSERT data structure. */
```

CM_INSERTRECORD - Remarks

The *pRecordInsert* parameter is used to insert [RECORDCORE](#) structures into the container. The *pRecordOrder* and *pRecordParent* fields of the [RECORDINSERT](#) data structure are used to place each record into the container in order, relative to the other records. If the [CMA_FIRST](#) or [CMA_END](#) attributes are specified, records are inserted before the first child or after the last child of the record specified in the *pRecordParent* field. If the value of the *pRecordParent* field is NULL, the record or records are inserted before the first record or after the last record, respectively, at the root level. Otherwise, if the value of the *pRecordOrder* field is a pointer to a record, the record or records to be inserted are placed after this record.

A z-ordering of the records is maintained by the container control. The *zOrder* field of the [RECORDINSERT](#) data structure is used to specify the record's z-order in the container, relative to the other records. The [CMA_TOP](#) attribute is used to place the record at the end of the z-order list, while the [CMA_BOTTOM](#) attribute places the record at the beginning of the z-order list. Z-ordering is used for the icon view only.

If the value of the *cRecordsInsert* field of the [RECORDINSERT](#) data structure is greater than 1, a linked list of [RECORDCORE](#) structures is inserted in the order specified by the *pRecordOrder*, *pRecordParent*, and *zOrder* fields. Here, the *pRecord* parameter points to the first [RECORDCORE](#) structure of a linked list of structures.

If one [RECORDCORE](#) structure is to be inserted, the *cRecordsInsert* field has a value of 1 and the *pRecord* parameter points to the [RECORDCORE](#) structure to be inserted.

When containers display the icon view, the coordinates specified by the [RECORDCORE](#) structure's *ptIcon* field are used to position inserted container records in the container's workspace. If the coordinates are not specified and the [CCS_AUTOPOSITION](#) style bit is not set, all of the inserted container records are positioned at (0,0) and a [CM_ARRANGE](#) message must be sent to position them elsewhere. If the [CCS_AUTOPOSITION](#) style bit is set, the container records are positioned without the [CM_ARRANGE](#) message being sent.

After the container records have been inserted:

- If the *flInvalidateRecord* field of the [RECORDINSERT](#) data structure is FALSE, the [CM_INVALIDATERECORD](#) message must be sent to update the display with the inserted records. If the current view is the icon view and either the CCS_AUTOPOSITION style bit is set or the *flInvalidateRecord* field is TRUE, the view is updated without the [CM_INVALIDATERECORD](#) message being sent.
- The *preccNextRecord*, *flRecordAttr*, and *ptllcon* fields of the external [RECORDCORE](#) structure are not updated as changes occur within the container. However, if records are shared among multiple containers, the *flRecordAttr* and *ptllcon* fields are modified internally. See the *PM Programming Guide* for more information about the modification of these fields.

If the CCS_VERIFYPOINTERS style bit is set and the *pRecord* parameter contains a pointer to a [RECORDCORE](#) structure that is currently inserted, the PMERR_RECORD_CURRENTLY_INSERTED error is set and no [RECORDCORE](#) structures are inserted.

If the [RECORDCORE](#) structures are sorted on insertion, the *pRecordOrder* and *zOrder* fields are ignored.

CM_INSERTRECORD - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

CM_INSERTRECORD - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CM_INSERTRECORDARRAY

CM_INSERTRECORDARRAY Field - pRecordArray

pRecordArray ([PVOID](#))

Pointer to an array of pointers to [RECORDCORE](#) structures that are to be inserted into the container.

CM_INSERTRECORDARRAY Field - pRecordInsert

pRecordInsert ([PRECORDINSERT](#))

Pointer to the [RECORDINSERT](#) structure.

CM_INSERTRECORDARRAY Return Value - cRecords

cRecords (ULONG)

Number of RECORDCORE structures in the root level of the container.

0

No RECORDCORE structures were inserted.

The WinGetLastError function may return the following errors:

PMERR_INVALID_PARAMETERS
PMERR_INSUFFICIENT_MEMORY
PMERR_RECORD_CURRENTLY_INSERTED

Other

The number of RECORDCORE structures in the container.

CM_INSERTRECORDARRAY - Parameters

pRecordArray (PVOID)

Pointer to an array of pointers to RECORDCORE structures that are to be inserted into the container.

pRecordInsert (PRECORDINSERT)

Pointer to the RECORDINSERT structure.

cRecords (ULONG)

Number of RECORDCORE structures in the root level of the container.

0

No RECORDCORE structures were inserted.

The WinGetLastError function may return the following errors:

PMERR_INVALID_PARAMETERS
PMERR_INSUFFICIENT_MEMORY
PMERR_RECORD_CURRENTLY_INSERTED

Other

The number of RECORDCORE structures in the container.

CM_INSERTRECORDARRAY - Syntax

This message inserts one or more RECORDCORE structures into a container control.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container control is created, then MINIRECORDCORE should be used instead of RECORDCORE, and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

param1

```

        PVOID          pRecordArray /* Pointer to an array of pointers to RECORDCORE structures that are to be
param2
        PRECORDINSERT pRecordInsert /* Pointer to the RECORDINSERT structure. */

```

CM_INSERTRECORDARRAY - Remarks

The *pRecordInsert* parameter is used to insert [RECORDCORE](#) structures into the container. The *pRecordOrder* and *pRecordParent* fields of the [RECORDINSERT](#) data structure are used to place each record into the container in order, relative to the other records. If the [CMA_FIRST](#) or [CMA_END](#) attributes are specified, records are inserted before the first child or after the last child of the record specified in the *pRecordParent* field. If the value of the *pRecordParent* field is NULL, the record or records are inserted before the first record or after the last record, respectively, at the root level. Otherwise, if the value of the *pRecordOrder* field is a pointer to a record, the record or records to be inserted are placed after this record.

A z-ordering of the records is maintained by the container control. The *zOrder* field of the [RECORDINSERT](#) data structure is used to specify the record's z-order in the container, relative to the other records. The [CMA_TOP](#) attribute is used to place the record at the end of the z-order list, while the [CMA_BOTTOM](#) attribute places the record at the beginning of the z-order list. Z-ordering is used for the icon view only.

The *cRecords* parameter always specifies an array of pointers to [RECORDCORE](#) structures to be inserted into the container. The number of pointers contained in the array must equal the value specified in the *cRecords/Insert* field of the [RECORDINSERT](#) structure.

When containers display the icon view, the coordinates specified by the [RECORDCORE](#) structure's *ptllcon* field are used to position inserted container records in the container's workspace. If the coordinates are not specified and the [CCS_AUTOPOSITION](#) style bit is not set, all of the inserted container records are positioned at (0,0) and a [CM_ARRANGE](#) message must be sent to position them elsewhere. If the [CCS_AUTOPOSITION](#) style bit is set, the container records are positioned without the [CM_ARRANGE](#) message being sent.

After the container records have been inserted:

- If the *flInvalidateRecord* field of the [RECORDINSERT](#) data structure is FALSE, the [CM_INVALIDATERECORD](#) message must be sent to update the display with the inserted records. If the current view is the icon view and either the [CCS_AUTOPOSITION](#) style bit is set or the *flInvalidateRecord* field is TRUE, the view is updated without the [CM_INVALIDATERECORD](#) message being sent.
- The *preccNextRecord*, *flRecordAttr*, and *ptllcon* fields of the external [RECORDCORE](#) structure are not updated as changes occur within the container. However, if records are shared among multiple containers, the *flRecordAttr* and *ptllcon* fields are modified internally.

If the [CCS_VERIFYPOINTERS](#) style bit is set and the *pRecordArray* parameter contains a pointer to a [RECORDCORE](#) structure that is currently inserted, the [PMERR_RECORD_CURRENTLY_INSERTED](#) error is set and no [RECORDCORE](#) structures are inserted.

If the [RECORDCORE](#) structures are sorted on insertion, the *pRecordOrder* and *zOrder* fields are ignored.

CM_INSERTRECORDARRAY - Default Processing

The default window procedure does not expect to receive this message and, therefore, takes no action on it other than to return FALSE.

CM_INSERTRECORDARRAY - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CM_INVALIDATEDDETAILFIELDINFO

CM_INVALIDATEDDETAILFIELDINFO Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

CM_INVALIDATEDDETAILFIELDINFO Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

CM_INVALIDATEDDETAILFIELDINFO Return Value - rc

rc (BOOL)
Success indicator.

TRUE
FIELDINFO structures were successfully refreshed.

FALSE
FIELDINFO structures were not successfully refreshed.

CM_INVALIDATEDDETAILFIELDINFO - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

rc (BOOL)
Success indicator.

TRUE
FIELDINFO structures were successfully refreshed.

FALSE

[FIELDINFO](#) structures were not successfully refreshed.

CM_INVALIDATEDDETAILFIELDINFO - Syntax

This message notifies the container control that any or all [FIELDINFO](#) structures are not valid and that the view must be refreshed.

```
param1
    ULONG  ulReserved /* Reserved value, should be 0. */

param2
    ULONG  ulReserved /* Reserved value, should be 0. */
```

CM_INVALIDATEDDETAILFIELDINFO - Remarks

If any or all [FIELDINFO](#) structures are changed, removed, or inserted, the CM_INVALIDATEDDETAILFIELDINFO message must be sent. Since each [FIELDINFO](#) structure potentially affects every record in the container, the entire view is refreshed, even if only one [FIELDINFO](#) structure has changed.

CM_INVALIDATEDDETAILFIELDINFO - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_INVALIDATEDDETAILFIELDINFO - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CM_INVALIDATERECORD

CM_INVALIDATERECORD Field - pRecordArray

pRecordArray ([PVOID](#))

Pointer to an array of pointers to [RECORDCORE](#) structures that are to be refreshed.

CM_INVALIDATERECORD Field - cNumRecord

cNumRecord ([USHORT](#))

Number of container records to be refreshed.

If the *cNumRecord* parameter has a value of 0, all of the records in the container are refreshed and the *pRecordArray* parameter is ignored.

CM_INVALIDATERECORD Field - flInvalidateRecord

flInvalidateRecord ([USHORT](#))

Flags used to optimize container record invalidation.

The CMA_REPOSITION, CMA_NOREPOSITION, and CMA_TEXTCHANGED attributes are mutually exclusive. However, any of them can be combined with the CMA_ERASE attribute by using a logical OR operator (|).

CMA_ERASE

Flag used when the icon view is displayed to minimize painting of a container record's background when it has changed. If specified, the background is erased when the display is refreshed. The default is to not erase the background when the display is refreshed.

CMA_REPOSITION

Flag used to reposition all container records. This flag must be used if container records are inserted or removed, or if many changes have occurred. If a container record is inserted, the *pRecordArray* parameter points to the inserted record. If a container record is removed, the *pRecordArray* parameter points to the record that precedes the removed one. If several container records have changed, an array of container record pointers must be used. The container determines the first record to be invalidated. This is the default.

CMA_NOREPOSITION

Flag used to indicate that container records do not need to be repositioned. The container draws the record or records pointed to in the *pRecordArray* parameter. The container does not do any validation; therefore it is the application's responsibility to make sure repositioning is not needed or changing the longest text line is not necessary.

CMA_TEXTCHANGED

Flag used if text has changed and you do not know whether repositioning is needed. The container determines whether the longest line or the height of the record has changed. If so, the container repositions and redraws the necessary visible container records.

It may be necessary to reposition the container records if the number of lines of text has changed.

Warning: The application must send a CM_INVALIDATERECORD message if text changes. Otherwise, any further processing is unreliable.

CM_INVALIDATERECORD Return Value - rc

rc (BOOL)

Success indicator.

TRUE

Records were successfully refreshed.

FALSE

An error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_INSUFFICIENT_MEMORY.

CM_INVALIDATERECORD - Parameters

pRecordArray (PVOID)

Pointer to an array of pointers to [RECORDCORE](#) structures that are to be refreshed.

cNumRecord (USHORT)

Number of container records to be refreshed.

If the *cNumRecord* parameter has a value of 0, all of the records in the container are refreshed and the *pRecordArray* parameter is ignored.

flInvalidateRecord (USHORT)

Flags used to optimize container record invalidation.

The CMA_REPOSITION, CMA_NOREPOSITION, and CMA_TEXTCHANGED attributes are mutually exclusive. However, any of them can be combined with the CMA_ERASE attribute by using a logical OR operator (|).

CMA_ERASE

Flag used when the icon view is displayed to minimize painting of a container record's background when it has changed. If specified, the background is erased when the display is refreshed. The default is to not erase the background when the display is refreshed.

CMA_REPOSITION

Flag used to reposition all container records. This flag must be used if container records are inserted or removed, or if many changes have occurred. If a container record is inserted, the *pRecordArray* parameter points to the inserted record. If a container record is removed, the *pRecordArray* parameter points to the record that precedes the removed one. If several container records have changed, an array of container record pointers must be used. The container determines the first record to be invalidated. This is the default.

CMA_NOREPOSITION

Flag used to indicate that container records do not need to be repositioned. The container draws the record or records pointed to in the *pRecordArray* parameter. The container does not do any validation; therefore it is the application's responsibility to make sure repositioning is not needed or changing the longest text line is not necessary.

CMA_TEXTCHANGED

Flag used if text has changed and you do not know whether repositioning is needed. The container determines whether the longest line or the height of the record has changed. If so, the container repositions and redraws the necessary visible container records.

It may be necessary to reposition the container records if the number of lines of text has changed.

Warning: The application must send a CM_INVALIDATERECORD message if text changes. Otherwise, any further processing is unreliable.

rc (BOOL)

Success indicator.

TRUE

Records were successfully refreshed.

FALSE

An error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_INSUFFICIENT_MEMORY.

CM_INVALIDATERECORD - Syntax

This message notifies the container control that a [RECORDCORE](#) structure or structures are not valid and must be refreshed.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

```
param1
    PVOID    pRecordArray    /* Pointer to an array of pointers to RECORDCORE structures that are to be ref
```

param2

```
    USHORT   cNumRecord      /* Number of container records to be refreshed. */
    USHORT   fInvalidateRecord /* Flags used to optimize container record invalidation. */
```

CM_INVALIDATERECORD - Remarks

If the number of pointers to container records in the array exceeds the count of records to be refreshed, only the number of records specified in the *cNumRecord* parameter is refreshed. If the CCS_VERIFYPOINTERS style bit is set and the *pRecordArray* parameter contains pointers to a [RECORDCORE](#) structure or structures that do not exist, the PMERR_INVALID_PARAMETERS error is set and nothing is refreshed.

CM_INVALIDATERECORD - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_INVALIDATERECORD - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CM_MOVETREE

CM_MOVETREE Field - pTreeMove

pTreeMove ([PTREEMOVE](#))
Pointer to a [TREEMOVE](#) structure.

See [TREEMOVE](#) for definitions of this structure's fields as they apply to the CM_MOVETREE message.

CM_MOVETREE Field - Reserved

Reserved ([ULONG](#))
Reserved value, must be 0.

CM_MOVETREE Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE Record and associated subtrees were moved successfully.
FALSE Error occurred, and tree structure remains unchanged.

CM_MOVETREE - Parameters

pTreeMove ([PTREEMOVE](#))
Pointer to a [TREEMOVE](#) structure.

See [TREEMOVE](#) for definitions of this structure's fields as they apply to the CM_MOVETREE message.

Reserved ([ULONG](#))
Reserved value, must be 0.

rc ([BOOL](#))
Success indicator.

TRUE

FALSE	Record and associated subtrees were moved successfully.
	Error occurred, and tree structure remains unchanged.

CM_MOVETREE - Syntax

This message is used to move a record to a new parent in the container control.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container control is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

```
param1
    PTREEMOVE  pTreeMove /* Pointer to a TREEMOVE structure. */

param2
    ULONG      Reserved /* Reserved value, must be 0. */
```

CM_MOVETREE - Remarks

This message is used to change the parent of a record in the container control. The fields of the [TREEMOVE](#) structure describe the record to be moved, the record to become its new parent, and where to insert the record relative to other records with the same parent.

If the *preccNewParent* field of the [TREEMOVE](#) structure is NULL, the record being moved is moved to the root level; otherwise, it is moved to *preccNewParent*. The *pRecordOrder* field of the [TREEMOVE](#) structure determines where the record being moved is placed relative to other records with the same parent (the one specified by *preccNewParent*). If *filMoveSiblings* of the [TREEMOVE](#) structure is TRUE, all siblings that follow the record being moved (*preccMove*) are moved to the new parent as well. Siblings that precede *preccMove* are not moved regardless of the value of the *filMoveSiblings* field. For normal Tree Move operations, the *filMoveSiblings* field of the [TREEMOVE](#) structure should be set to FALSE.

[WinGetLastError](#) returns PMERR_INVALID_PARAMETERS if any of the following illegal combinations are used:

- *filMoveSiblings* is either the first or last root level record in the container, and the *filMoveSiblings* flag is TRUE.
- *preccMove* is a root level record, and *preccNewParent* is currently one of its children.
- *pRecordOrder* is a pointer to a [RECORDCORE](#) structure (not CMA_FIRST or CMA_LAST) that does not exist in the list of children of the new parent.
- *preccNewParent* is NULL, and *pRecordOrder* is not a root level record.

For example, the following tree contains two parents, each with three children:

```
Parent A
    Child A1
    Child A2
    Child A3

Parent B
    Child B1
    Child B2
```

Child B3

If *preccMove* is Child A2, *preccNewParent* is Parent B, *pRecordOrder* = CMA_LAST and *//MoveSiblings* = TRUE, after the Tree Move operation, the new tree structure is as follows:

Parent A

Child A1

Parent B

Child B1

Child B2

Child B3

Child A2

Child A3

CM_MOVETREE - Default Processing

The default window procedure does not expect to receive this message and, therefore, takes no action on it other than to return FALSE.

CM_MOVETREE - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

CM_OPENEDIT

CM_OPENEDIT Field - pCnrEditData

pCnrEditData ([PCNREDITDATA](#))

Pointer to the [CNREDITDATA](#) structure.

CM_OPENEDIT Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

CM_OPENEDIT Return Value - rc

rc (BOOL)
Success indicator.

TRUE
Direct editing of container text was successfully started.

FALSE
Direct editing of container text was not successfully started. The [WinGetLastError](#) function may return the following error:
PMERR_INVALID_PARAMETERS.

CM_OPENEDIT - Parameters

pCnrEditData (PCNREDITDATA)
Pointer to the [CNREDITDATA](#) structure.

ulReserved (ULONG)
Reserved value, should be 0.

rc (BOOL)
Success indicator.

TRUE
Direct editing of container text was successfully started.

FALSE
Direct editing of container text was not successfully started. The [WinGetLastError](#) function may return the following error:
PMERR_INVALID_PARAMETERS.

CM_OPENEDIT - Syntax

This message opens the window that contains the multiple-line entry (MLE) field used to edit container text directly.

```
param1
    PCNREDITDATA pCnrEditData /* Pointer to the CNREDITDATA structure. */

param2
    ULONG        ulReserved    /* Reserved value, should be 0. */
```

CM_OPENEDIT - Remarks

The application sends this message to the container control to start the direct editing of container text. The application can assign this message to a key or key combination, a menu choice, or both so that the user can start editing container text directly from the keyboard.

When the container control receives this message, it sends the [CN_BEGINEDIT](#) notification code to the application.

CM_OPENEDIT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_OPENEDIT - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CM_PAINTBACKGROUND

CM_PAINTBACKGROUND Field - pOwnerBackground

pOwnerBackground ([POWNERBACKGROUND](#))
Pointer to the [OWNERBACKGROUND](#) structure.

CM_PAINTBACKGROUND Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

CM_PAINTBACKGROUND Return Value - rc

rc (BOOL)

Process indicator.

TRUE

The application processed the CM_PAINTBACKGROUND message.

FALSE

The application did not process the CM_PAINTBACKGROUND message.

CM_PAINTBACKGROUND - Parameters

pOwnerBackground (POWNERBACKGROUND)

Pointer to the OWNERBACKGROUND structure.

ulReserved (ULONG)

Reserved value, should be 0.

rc (BOOL)

Process indicator.

TRUE

The application processed the CM_PAINTBACKGROUND message.

FALSE

The application did not process the CM_PAINTBACKGROUND message.

CM_PAINTBACKGROUND - Syntax

This message informs an application whenever a container's background is painted if the CA_OWNERPAINTBACKGROUND attribute of the CNRINFO data structure is specified.

```
param1
    POWNERBACKGROUND pOwnerBackground /* Pointer to the OWNERBACKGROUND structure. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

CM_PAINTBACKGROUND - Remarks

The CM_PAINTBACKGROUND message is provided so that an application can subclass the container control and paint its own background. If the application does not subclass the container control or subclasses the container control and returns FALSE, the container uses the system window color, which is specified by SYSCLR_WINDOW. This color can be changed by using the PP_BACKGROUND_COLOR or PP_BACKGROUND_COLOR_INDEX presentation parameter of the WM_PRESPARAMCHANGED (in Container Controls) message.

CM_PAINTBACKGROUND - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_PAINTBACKGROUND - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

CM_QUERYCNRINFO

CM_QUERYCNRINFO Field - pCnrInfo

pCnrInfo ([PCNRINFO](#))
Pointer to a buffer into which the [CNRINFO](#) structure is copied.

CM_QUERYCNRINFO Field - cbBuffer

cbBuffer ([USHORT](#))
Number of bytes.

Maximum number of bytes to copy.

CM_QUERYCNRINFO Return Value - cbBytes

cbBytes ([USHORT](#))
Success indicator.

0	Container data was not successfully returned. The WinGetLastError function may return the following error: PMERR_INVALID_PARAMETERS.
Other	Actual number of bytes copied.

CM_QUERYCNRINFO - Parameters

pCnrInfo ([PCNRINFO](#))
Pointer to a buffer into which the [CNRINFO](#) structure is copied.

cbBuffer ([USHORT](#))
Number of bytes.

Maximum number of bytes to copy.

cbBytes ([USHORT](#))
Success indicator.

0	Container data was not successfully returned. The WinGetLastError function may return the following error: PMERR_INVALID_PARAMETERS.
Other	Actual number of bytes copied.

CM_QUERYCNRINFO - Syntax

This message returns the container's [CNRINFO](#) structure.

```
param1
    PCNRINFO  pCnrInfo /* Pointer to a buffer into which the CNRINFO structure is copied. */
param2
    USHORT    cbBuffer /* Number of bytes. */
```

CM_QUERYCNRINFO - Remarks

The number of bytes specified in the *cbBuffer* parameter is returned in the buffer addressed by the *pCnrInfo* parameter.

CM_QUERYCNRINFO - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

CM_QUERYCNRINFO - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

CM_QUERYDETAILFIELDINFO

CM_QUERYDETAILFIELDINFO Field - pfldinfoBase

pfldinfoBase ([PFIELDINFO](#))
Pointer to the [FIELDINFO](#) structure used to search for the next or previous column.
If the CMA_FIRST or CMA_LAST attribute is specified, this is ignored.

CM_QUERYDETAILFIELDINFO Field - cmd

cmd ([USHORT](#))
Command that indicates which [FIELDINFO](#) structure to retrieve.

CMA_FIRST	First column in the container.
CMA_LAST	Last column in the container.
CMA_NEXT	Next column in the container.
CMA_PREV	Previous column in the container.

CM_QUERYDETAILFIELDINFO Return Value - pFieldInfo

pFieldInfo (PFIELDINFO)	Pointer to the FIELDINFO structure for which data was requested.
NULL	No FIELDINFO structures to retrieve.
-1	The data from the FIELDINFO structure was not returned. The WinGetLastError function may return the following error: PMERR_INVALID_PARAMETERS.
Other	Pointer to the FIELDINFO structure for which data was requested.

CM_QUERYDETAILFIELDINFO - Parameters

pfldinfoBase (PFIELDINFO)	Pointer to the FIELDINFO structure used to search for the next or previous column. If the CMA_FIRST or CMA_LAST attribute is specified, this is ignored.
cmd (USHORT)	Command that indicates which FIELDINFO structure to retrieve. CMA_FIRST First column in the container. CMA_LAST Last column in the container. CMA_NEXT Next column in the container. CMA_PREV Previous column in the container.
pFieldInfo (PFIELDINFO)	Pointer to the FIELDINFO structure for which data was requested. NULL No FIELDINFO structures to retrieve. -1 The data from the FIELDINFO structure was not returned. The WinGetLastError function may return the following error: PMERR_INVALID_PARAMETERS. Other Pointer to the FIELDINFO structure for which data was requested.

CM_QUERYDETAILFIELDINFO - Syntax

This message returns a pointer to the requested **FIELDINFO** structure.

```
param1
    PFIELDINFO  pfldinfoBase  /*  Pointer to the FIELDINFO structure used to search for the next or previous c
```

param2
USHORT cmd /* Command that indicates which FIELDINFO structure to retrieve. */

CM_QUERYDETAILFIELDINFO - Remarks

If the *cmd* parameter has the value of the CMA_FIRST or CMA_LAST attribute, the *pFldInfoBase* parameter is ignored and the first or last column data, respectively, is returned. If the CMA_NEXT or the CMA_PREV attribute is set in the *cmd* parameter, the column data next to or before the column pointed to by the *pFieldInfo* parameter is returned.

CM_QUERYDETAILFIELDINFO - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return NULL.

CM_QUERYDETAILFIELDINFO - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

CM_QUERYDRAGIMAGE

CM_QUERYDRAGIMAGE Field - pRecord

pRecord (PRECORDCORE)
Pointer to the RECORDCORE structure that is to be queried for the image.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

CM_QUERYDRAGIMAGE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

CM_QUERYDRAGIMAGE Return Value - hImage

hImage ([LHANDLE](#))
Image handle.

NULLHANDLE
If no image is defined, NULLHANDLE is returned.

Other
Handle of an icon or bit map.

- If the CA_DRAWICON attribute and the CV_MINI style bit are specified, the [RECORDCORE](#) structure's *hptrMiniIcon* field is returned.
- If the CA_DRAWICON attribute is specified without the CV_MINI style bit, the [RECORDCORE](#) structure's *hptrIcon* field is returned.
- If the CA_DRAWBITMAP attribute and the CV_MINI style bit are specified, the [RECORDCORE](#) structure's *hbmMiniBitmap* field is returned.
- If the CA_DRAWBITMAP attribute is specified without the CV_MINI style bit, the [RECORDCORE](#) structure's *hbmBitmap* field is returned.

CM_QUERYDRAGIMAGE - Parameters

pRecord ([PRECORDCORE](#))
Pointer to the [RECORDCORE](#) structure that is to be queried for the image.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

ulReserved ([ULONG](#))
Reserved value, should be 0.

hImage ([LHANDLE](#))
Image handle.

NULLHANDLE
If no image is defined, NULLHANDLE is returned.

Other
Handle of an icon or bit map.

- If the CA_DRAWICON attribute and the CV_MINI style bit are specified, the [RECORDCORE](#) structure's *hptrMiniIcon* field is returned.
- If the CA_DRAWICON attribute is specified without the CV_MINI style bit, the [RECORDCORE](#) structure's *hptrIcon* field is returned.
- If the CA_DRAWBITMAP attribute and the CV_MINI style bit are specified, the [RECORDCORE](#)

structure's *hbmMiniBitmap* field is returned.

- If the CA_DRAWBITMAP attribute is specified without the CV_MINI style bit, the [RECORDCORE](#) structure's *hbmBitmap* field is returned.

CM_QUERYDRAGIMAGE - Syntax

This message returns a handle to the icon or bit map for the record in the current view.

```
param1
    PRECORDCORE pRecord    /* Pointer to the RECORDCORE structure that is to be queried for the image. */
param2
    ULONG          ulReserved /* Reserved value, should be 0. */
```

CM_QUERYDRAGIMAGE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return NULLHANDLE.

CM_QUERYDRAGIMAGE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Glossary](#)

CM_QUERYGRIDINFO

CM_QUERYGRIDINFO Field - PGRIDINFO

PGRIDINFO ([PGRIDINFO](#))
Pointer to a GRIDINFO structure.

CM_QUERYGRIDINFO Field - ulReserved

ulReserved (ULONG)
This value is NULL.

CM_QUERYGRIDINFO Return Value - rc

rc (BOOL)
Success indicator.

TRUE	The GRIDINFO structure has been updated with the grid information.
FALSE	An error has occurred.

CM_QUERYGRIDINFO - Parameters

PGRIDINFO (PGRIDINFO)
Pointer to a GRIDINFO structure.

ulReserved (ULONG)
This value is NULL.

rc (BOOL)
Success indicator.

TRUE	The GRIDINFO structure has been updated with the grid information.
FALSE	An error has occurred.

CM_QUERYGRIDINFO - Syntax

This message queries the current characteristics of the grid.

```
param1
    PGRIDINFO  PGRIDINFO  /*  Pointer to a GRIDINFO structure. */

param2
    ULONG      ulReserved /*  This value is NULL. */
```

CM_QUERYGRIDINFO - Remarks

The *cb* and *pGrid* fields can be set by the application with CM_SETGRIDINFO before sending this message.

If *pGrid* is NULL, the fields *cxGrid*, *cyGrid*, *sGridRows*, *sGridCols*, and *cGridSquares* are updated by the container.

If *pGrid* is non-NULL, the container expects it to point to an array of GRIDSQUARE structures whose size is equal to *cGridSquares**sizeof(GRIDSQUARE). The container will copy its internal grid square information into the GRIDINFO structure passed by the application.

CM_QUERYGRIDINFO - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_QUERYGRIDINFO - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

CM_QUERYRECORD

CM_QUERYRECORD Field - pRecord

pRecord ([PRECORDCORE](#))

Pointer to the [RECORDCORE](#) structure used to search for the next or previous container record.

If the CMA_FIRST or CMA_LAST attribute is specified, this is ignored.

CM_QUERYRECORD Field - cmd

cmd (USHORT)	Command that indicates which container record to retrieve:
CMA_FIRST	First record in the container.
CMA_FIRSTCHILD	First child record of <i>pRecord</i> specified in <i>param1</i> .
CMA_LAST	Last record in the container.
CMA_LASTCHILD	Last child record of <i>pRecord</i> specified in <i>param1</i> .
CMA_NEXT	Next record of <i>pRecord</i> specified in <i>param1</i> .
CMA_PARENT	Parent of <i>pRecord</i> specified in <i>param1</i> .
CMA_PREV	Previous record of <i>pRecord</i> specified in <i>param1</i> .

CM_QUERYRECORD Field - fsSearch

fsSearch (USHORT)	Enumeration order.
	Specifies the enumeration order. This value is one of the following:
CMA_ITEMORDER	Container records are enumerated in item order, first to last.
CMA_ZORDER	Container records are enumerated by z-order, from first record in the z-order to the last record. The last z-order record is the last record to be drawn. This flag is valid for the icon view only.

CM_QUERYRECORD Return Value - pRecord

pRecord (PRECORDCORE)	Pointer to the RECORDCORE structure for which data was requested.
NULL	No RECORDCORE structures to retrieve.
-1	The container record data was not returned. The WinGetLastError function may return the following error: PMERR_INVALID_PARAMETERS.
Other	Pointer to the container record for which data was requested.

CM_QUERYRECORD - Parameters

pRecord (PRECORDCORE)
--

Pointer to the [RECORDCORE](#) structure used to search for the next or previous container record.

If the CMA_FIRST or CMA_LAST attribute is specified, this is ignored.

cmd ([USHORT](#))

Command that indicates which container record to retrieve:

CMA_FIRST	First record in the container.
CMA_FIRSTCHILD	First child record of <i>pRecord</i> specified in <i>param1</i> .
CMA_LAST	Last record in the container.
CMA_LASTCHILD	Last child record of <i>pRecord</i> specified in <i>param1</i> .
CMA_NEXT	Next record of <i>pRecord</i> specified in <i>param1</i> .
CMA_PARENT	Parent of <i>pRecord</i> specified in <i>param1</i> .
CMA_PREV	Previous record of <i>pRecord</i> specified in <i>param1</i> .

fsSearch ([USHORT](#))

Enumeration order.

Specifies the enumeration order. This value is one of the following:

CMA_ITEMORDER	Container records are enumerated in item order, first to last.
CMA_ZORDER	Container records are enumerated by z-order, from first record in the z-order to the last record. The last z-order record is the last record to be drawn. This flag is valid for the icon view only.

pRecord ([PRECORDCORE](#))

Pointer to the [RECORDCORE](#) structure for which data was requested.

NULL	No RECORDCORE structures to retrieve.
-1	The container record data was not returned. The WinGetLastError function may return the following error: PMERR_INVALID_PARAMETERS.
Other	Pointer to the container record for which data was requested.

CM_QUERYRECORD - Syntax

This message returns a pointer to the requested [RECORDCORE](#) structure.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

```
param1
    PRECORDCORE    pRecord    /* Pointer to the RECORDCORE structure used to search for the next or previous con

param2
    USHORT          cmd         /* Command that indicates which container record to retrieve: */
    USHORT          fsSearch    /* Enumeration order. */
```

CM_QUERYRECORD - Remarks

If the *cmd* parameter has the value of CMA_FIRST or CMA_LAST, the *pRecord* parameter in *param1* is ignored and the first or last record, respectively, in the container is returned.

Depending on the value of the *fsSearch* parameter, the container records are enumerated in item order or in z-order.

See [RECORDCORE](#) or [MINIRECORDCORE](#) for a complete list and descriptions of all container record attributes.

CM_QUERYRECORD - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return NULL.

CM_QUERYRECORD - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

CM_QUERYRECORDEMPHASIS

CM_QUERYRECORDEMPHASIS Field - pSearchAfter

pSearchAfter ([PRECORDCORE](#))

Pointer to the specified container record.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

The values of this parameter can be:

CMA_FIRST

Start the search with the first record in the container.

Other

Start the search after the record specified by this pointer.

CM_QUERYRECORDEMPHASIS Field - fEmphasisMask

fEmphasisMask ([USHORT](#))
Emphasis attribute.

Specifies the emphasis attribute of the container record. The following states can be combined using a logical OR operator (|):

CRA_COLLAPSED	Specifies that a record is collapsed.
CRA_CURSORED	Specifies that a record will be drawn with a selection cursor.
CRA_DISABLED	Specifies that a record will be drawn with unavailable-state emphasis.
CRA_DROPONABLE	Specifies that a record can be a target for direct manipulation.
CRA_EXPANDED	Specifies that a record is expanded.
CRA_FILTERED	Specifies that a record is filtered and, therefore, hidden from view.
CRA_INUSE	Specifies that a record will be drawn with in-use emphasis.
CRA_PICKED	Specifies that the container record will be picked up as part of the drag set.
CRA_SELECTED	Specifies that a record will be drawn with selected-state emphasis.
CRA_SOURCE	Specifies that a record will be drawn with source-menu emphasis.

CM_QUERYRECORDEMPHASIS Return Value - pRecord

pRecord ([PRECORDCORE](#))
Pointer to the record with the specified emphasis.

NULL	This implies that none of the records that follow the pointer specified in the <i>pSearchAfter</i> parameter meet those specifications.
-1	The container record data was not returned. The WinGetLastError function may return the following error: PMERR_INVALID_PARAMETERS (1208)
Other	Pointer to a container record with the specified emphasis. This is the first record that follows the record pointed to by the <i>pSearchAfter</i> parameter and satisfies the criteria specified in the <i>fEmphasisMask</i> parameter. To find the next record that satisfies this criteria, send this message again, but this time use the value returned in the <i>pRecord</i> parameter for the value of the <i>pSearchAfter</i> parameter.

CM_QUERYRECORDEMPHASIS - Parameters

pSearchAfter ([PRECORDCORE](#))
Pointer to the specified container record.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

The values of this parameter can be:

- | | |
|-----------|--|
| CMA_FIRST | Start the search with the first record in the container. |
| Other | Start the search after the record specified by this pointer. |

fEmphasisMask ([USHORT](#))
Emphasis attribute.

Specifies the emphasis attribute of the container record. The following states can be combined using a logical OR operator (|):

- | | |
|----------------|--|
| CRA_COLLAPSED | Specifies that a record is collapsed. |
| CRA_CURSORED | Specifies that a record will be drawn with a selection cursor. |
| CRA_DISABLED | Specifies that a record will be drawn with unavailable-state emphasis. |
| CRA_DROPONABLE | Specifies that a record can be a target for direct manipulation. |
| CRA_EXPANDED | Specifies that a record is expanded. |
| CRA_FILTERED | Specifies that a record is filtered and, therefore, hidden from view. |
| CRA_INUSE | Specifies that a record will be drawn with in-use emphasis. |
| CRA_PICKED | Specifies that the container record will be picked up as part of the drag set. |
| CRA_SELECTED | Specifies that a record will be drawn with selected-state emphasis. |
| CRA_SOURCE | Specifies that a record will be drawn with source-menu emphasis. |

pRecord ([PRECORDCORE](#))
Pointer to the record with the specified emphasis.

- | | |
|-------|--|
| NULL | This implies that none of the records that follow the pointer specified in the <i>pSearchAfter</i> parameter meet those specifications. |
| -1 | The container record data was not returned.

The WinGetLastError function may return the following error:

PMERR_INVALID_PARAMETERS (1208) |
| Other | Pointer to a container record with the specified emphasis.

This is the first record that follows the record pointed to by the <i>pSearchAfter</i> parameter and satisfies the criteria specified in the <i>fEmphasisMask</i> parameter. To find the next record that satisfies this criteria, send this message again, but this time use the value returned in the <i>pRecord</i> parameter for the value of the <i>pSearchAfter</i> parameter. |

CM_QUERYRECORDEMPHASIS - Syntax

This message queries for a container record with the specified emphasis attributes.

```
param1
    PRECORDCORE pSearchAfter /* Pointer to the specified container record. */

param2
    USHORT      fEmphasisMask /* Emphasis attribute. */
```

CM_QUERYRECORDEMPHASIS - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return NULL.

CM_QUERYRECORDEMPHASIS - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Default Processing](#)

[Glossary](#)

CM_QUERYRECORDFROMRECT

CM_QUERYRECORDFROMRECT Field - pSearchAfter

pSearchAfter ([PRECORDCORE](#))

Pointer to the specified container record.

To get all the container records within the specified rectangle, this message is sent repeatedly, each time this parameter is set to the pointer that is returned by the previous usage of this message.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages. The values of this parameter can be:

CMA_FIRST

Start the search with the first record in the container.

Other

Start the search after the record specified by this pointer.

CM_QUERYRECORDFROMRECT Field -

pQueryRecFromRect

pQueryRecFromRect ([PQUERYRECFROMRECT](#))

Pointer to the [QUERYRECFROMRECT](#) data structure.

CM_QUERYRECORDFROMRECT Return Value - pRecord

pRecord ([PRECORDCORE](#))

Pointer to the container records within the bounding rectangle.

NULL

No container records are within the bounding rectangle.

-1

The container record data was not returned. The [WinGetLastError](#) function may return the following error:

PMERR_INVALID_PARAMETERS.

Other

Pointer to the container record within the bounding rectangle.

CM_QUERYRECORDFROMRECT - Parameters

pSearchAfter ([PRECORDCORE](#))

Pointer to the specified container record.

To get all the container records within the specified rectangle, this message is sent repeatedly, each time this parameter is set to the pointer that is returned by the previous usage of this message.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages. The values of this parameter can be:

CMA_FIRST

Start the search with the first record in the container.

Other

Start the search after the record specified by this pointer.

pQueryRecFromRect ([PQUERYRECFROMRECT](#))

Pointer to the [QUERYRECFROMRECT](#) data structure.

pRecord ([PRECORDCORE](#))

Pointer to the container records within the bounding rectangle.

NULL

No container records are within the bounding rectangle.

-1

The container record data was not returned. The [WinGetLastError](#) function may return the following error:

PMERR_INVALID_PARAMETERS.

Other

Pointer to the container record within the bounding rectangle.

CM_QUERYRECORDFROMRECT - Syntax

This message queries for a container record that is bounded by the specified rectangle.

```
param1
    RECORDCORE      pSearchAfter      /* Pointer to the specified container record. */
param2
    QUERYRECFROMRECT pQueryRecFromRect /* Pointer to the QUERYRECFROMRECT data structure. */
```

CM_QUERYRECORDFROMRECT - Remarks

This message returns the pointer to the first container record found in the rectangle after the starting position specified in the *pSearchAfter* parameter.

CM_QUERYRECORDFROMRECT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return NULL.

CM_QUERYRECORDFROMRECT - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CM_QUERYRECORDINFO

CM_QUERYRECORDINFO Field - pRecordArray

pRecordArray ([PVOID](#))

Pointer to an array of pointers to [RECORDCORE](#) structures to which the current information is to be copied.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE all applicable data structures and messages.

CM_QUERYRECORDINFO Field - cNumRecord

cNumRecord ([USHORT](#))

Number of records.

The number of container records to be updated. If the *cNumRecord* parameter has a value of 0, all of the records in the container are updated and the *pRecordArray* parameter is ignored.

CM_QUERYRECORDINFO Return Value - rc

rc ([BOOL](#))

Success indicator.

TRUE

Record information was successfully updated.

FALSE

An error occurred. The [WinGetLastError](#) function may return the following error:
PMERR_INVALID_PARAMETERS.

CM_QUERYRECORDINFO - Parameters

pRecordArray ([PVOID](#))

Pointer to an array of pointers to [RECORDCORE](#) structures to which the current information is to be copied.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE all applicable data structures and messages.

cNumRecord ([USHORT](#))

Number of records.

The number of container records to be updated. If the *cNumRecord* parameter has a value of 0, all of the records in the container are updated and the *pRecordArray* parameter is ignored.

rc ([BOOL](#))

Success indicator.

TRUE

Record information was successfully updated.

FALSE

An error occurred. The [WinGetLastError](#) function may return the following error:
PMERR_INVALID_PARAMETERS.

CM_QUERYRECORDINFO - Syntax

This message updates the specified records with the current information for the container.

```
param1
    PVOID    pRecordArray /* Pointer to an array of pointers to RECORDCORE structures to which the current in
param2
    USHORT   cNumRecord    /* Number of records. */
```

CM_QUERYRECORDINFO - Remarks

This message is needed only if the application is sharing records among multiple containers in the same process.

The *filRecordAttr* and *ptilcon* fields are updated internally when they change, but not in the external [RECORDCORE](#) structure. Therefore, the application's external record does not always have current information in these fields. This message is only needed if the application is sharing records among multiple containers in the same process.

CM_QUERYRECORDINFO - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_QUERYRECORDINFO - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CM_QUERYRECORDRECT

CM_QUERYRECORDRECT Field - prclItem

prclItem ([PRECTL](#))
Pointer to the [RECTL](#) structure, into which the rectangular coordinates are placed.

CM_QUERYRECORDRECT Field - pQueryRecordRect

pQueryRecordRect ([PQUERYRECORDRECT](#))
Pointer to the [QUERYRECORDRECT](#) structure.

CM_QUERYRECORDRECT Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE
A rectangle with valid coordinates is returned.

FALSE
The rectangle is not successfully returned. The [WinGetLastError](#) function may return the following error:
PMERR_INVALID_PARAMETERS.

CM_QUERYRECORDRECT - Parameters

prclItem ([PRECTL](#))
Pointer to the [RECTL](#) structure, into which the rectangular coordinates are placed.

pQueryRecordRect ([PQUERYRECORDRECT](#))
Pointer to the [QUERYRECORDRECT](#) structure.

rc ([BOOL](#))
Success indicator.

TRUE
A rectangle with valid coordinates is returned.

FALSE
The rectangle is not successfully returned. The [WinGetLastError](#) function may return the following error:
PMERR_INVALID_PARAMETERS.

CM_QUERYRECORDRECT - Syntax

This message returns the rectangle of the specified container record, relative to the container window origin.

```
param1
    PRECTL          prclItem          /* Pointer to the RECTL structure, into which the rectangular coordin
param2
    QUERYRECORDRECT pQueryRecordRect /* Pointer to the QUERYRECORDRECT structure. */
```

CM_QUERYRECORDRECT - Remarks

The coordinates of the returned rectangle are in window coordinates.

If the input record is not found in the container, the output rectangle is empty.

For a container using the details view (CV_DETAIL), all of the data for a row is returned in the rectangle.

CM_QUERYRECORDRECT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_QUERYRECORDRECT - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CM_QUERYVIEWPORTRECT

CM_QUERYVIEWPORTRECT Field - prclViewport

prclViewport ([PRECTL](#))
Pointer to the [RECTL](#) structure.

Pointer to the [RECTL](#) structure that the virtual coordinates of the client area rectangle are to be written into.

CM_QUERYVIEWPORTRECT Field - usIndicator

usIndicator ([USHORT](#))

Coordinate space indicator.

One of the following must be used:

CMA_WINDOW

Returns the client area rectangle in container window coordinates.

CMA_WORKSPACE

Return the client area rectangle in coordinates relative to the origin of the container's workspace.

CM_QUERYVIEWPORTRECT Field - fRightSplitWindow

fRightSplitWindow ([BOOL](#))

Flag.

Flag that specifies the right or left window in the split details view. This flag is ignored if the view is not the split details view.

TRUE

Right split window is returned.

FALSE

Left split window is returned.

CM_QUERYVIEWPORTRECT Return Value - rc

rc ([BOOL](#))

Success indicator.

TRUE

The client area rectangle was returned successfully.

FALSE

An error occurred. The [WinGetLastError](#) function may return the following error:

PMERR_INVALID_PARAMETERS.

CM_QUERYVIEWPORTRECT - Parameters

prclViewport ([PRECTL](#))

Pointer to the [RECTL](#) structure.

Pointer to the [RECTL](#) structure that the virtual coordinates of the client area rectangle are to be written into.

usIndicator ([USHORT](#))

Coordinate space indicator.

One of the following must be used:

CMA_WINDOW

Returns the client area rectangle in container window coordinates.

CMA_WORKSPACE

Return the client area rectangle in coordinates relative to the origin of the container's workspace.

fRightSplitWindow ([BOOL](#))

Flag.

Flag that specifies the right or left window in the split details view. This flag is ignored if the view is not the split details view.

TRUE

Right split window is returned.

FALSE

Left split window is returned.

rc ([BOOL](#))

Success indicator.

TRUE

The client area rectangle was returned successfully.

FALSE

An error occurred. The [WinGetLastError](#) function may return the following error:

PMERR_INVALID_PARAMETERS.

CM_QUERYVIEWPORTRECT - Syntax

This message returns a rectangle that contains the coordinates of the container's client area. These are virtual coordinates that are relative to the origin of the coordinate space requested.

```
param1
    PRECTL  prclViewport      /* Pointer to the RECTL structure. */

param2
    USHORT  usIndicator       /* Coordinate space indicator. */
    BOOL    fRightSplitWindow /* Flag. */
```

CM_QUERYVIEWPORTRECT - Remarks

The virtual coordinates of the client area rectangle are written into the structure addressed by the *prclViewport* parameter.

CM_QUERYVIEWPORTRECT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_QUERYVIEWPORTRECT - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CM_REMOVEDETAILEDINFO

CM_REMOVEDETAILEDINFO Field - pFieldInfoArray

pFieldInfoArray ([PVOID](#))

Pointer to an array of pointers to [FIELDINFO](#) structures that are to be removed.

CM_REMOVEDETAILEDINFO Field - cNumFieldInfo

cNumFieldInfo ([USHORT](#))

Number of [FIELDINFO](#) structures to be removed.

If the *cNumFieldInfo* parameter has a value of 0, all of the [FIELDINFO](#) structures in the container are removed and the *pFieldInfoArray* parameter is ignored.

CM_REMOVEDETAILEDINFO Field - fRemoveFieldInfo

fRemoveFieldInfo ([USHORT](#))

Flags.

Flags that show whether memory must be freed and [FIELDINFO](#) structures invalidated.

CMA_FREE

If specified, [FIELDINFO](#) structures are removed and memory associated with the [FIELDINFO](#) structures is freed. If not specified, [FIELDINFO](#) structures are removed and no memory is freed; this is the default.

CMA_INVALIDATE

If specified, after [FIELDINFO](#) structures are removed, the container is invalidated, and any necessary repositioning

of the [FIELDINFO](#) structures is performed. If not specified, invalidation is not performed.

CM_REMOVEDTAILFIELDINFO Return Value - cFields

cFields ([SHORT](#))

Number of structures.

-1

An error occurred. The [WinGetLastError](#) function may return the following errors:

- [PMERR_INVALID_PARAMETERS](#)
- [PMERR_MEMORY_DEALLOCATION_ERR](#).

Other

The number of [FIELDINFO](#) structures that remain in the container.

CM_REMOVEDetailFIELDINFO - Parameters

pFieldInfoArray ([PVOID](#))

Pointer to an array of pointers to [FIELDINFO](#) structures that are to be removed.

cNumFieldInfo ([USHORT](#))

Number of [FIELDINFO](#) structures to be removed.

If the *cNumFieldInfo* parameter has a value of 0, all of the [FIELDINFO](#) structures in the container are removed and the *pFieldInfoArray* parameter is ignored.

fRemoveFieldInfo ([USHORT](#))

Flags.

Flags that show whether memory must be freed and [FIELDINFO](#) structures invalidated.

CMA_FREE

If specified, [FIELDINFO](#) structures are removed and memory associated with the [FIELDINFO](#) structures is freed. If not specified, [FIELDINFO](#) structures are removed and no memory is freed; this is the default.

CMA_INVALIDATE

If specified, after [FIELDINFO](#) structures are removed, the container is invalidated, and any necessary repositioning of the [FIELDINFO](#) structures is performed. If not specified, invalidation is not performed.

cFields ([SHORT](#))

Number of structures.

-1

An error occurred. The [WinGetLastError](#) function may return the following errors:

- [PMERR_INVALID_PARAMETERS](#)
- [PMERR_MEMORY_DEALLOCATION_ERR](#).

Other

The number of [FIELDINFO](#) structures that remain in the container.

CM_REMOVEDetailFIELDINFO - Syntax

This message removes one, multiple, or all [FIELDINFO](#) structures from the container control.

```
param1
    PVOID    pFieldInfoArray    /* Pointer to an array of pointers to FIELDINFO structures that are to be removed. */

param2
    USHORT   cNumFieldInfo       /* Number of FIELDINFO structures to be removed. */
    USHORT   fRemoveFieldInfo    /* Flags. */
```

CM_REMOVEDETAILEDINFO - Remarks

The [FIELDINFO](#) structures are removed from the list of columns inserted into the container control.

If the `CMA_FREE` attribute is not specified, the container control removes the specified [FIELDINFO](#) structures without freeing the memory. The application is responsible for freeing the memory associated with the [FIELDINFO](#) structures by using the [CM_FREEDETAILEDINFO](#) message.

If the *cNumFieldInfo* parameter has a value of 0 and the `CMA_FREE` attribute is specified, all of the [FIELDINFO](#) structures in the container control are removed and the memory associated with the [FIELDINFO](#) structures is freed. It is the application's responsibility to free all of the application-allocated memory associated with the [FIELDINFO](#) structures.

If the number of pointers to [FIELDINFO](#) structures in the array exceeds the count of [FIELDINFO](#) structures to be removed, only the number of structures specified in the *cNumFieldInfo* parameter are removed. If the `CCS_VERIFYPOINTERS` style bit is set and the *pFieldInfoArray* parameter contains pointers to a [FIELDINFO](#) structure or structures that do not exist, the `PMERR_INVALID_PARAMETERS` error is set.

If you do not want to show a column, you can hide it by setting the `CFA_INVISIBLE` attribute of the [FIELDINFO](#) data structure and notifying the container control with the [CM_INVALIDATEDETAILEDINFO](#) message.

If the `CMA_INVALIDATE` attribute is specified, the container is repainted.

CM_REMOVEDETAILEDINFO - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

CM_REMOVEDETAILEDINFO - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CM_REMOVERECORD

CM_REMOVERECORD Field - pRecordArray

pRecordArray ([PVOID](#))

Pointer to an array of pointers to [RECORDCORE](#) structures that are to be removed.

CM_REMOVERECORD Field - cNumRecord

cNumRecord ([USHORT](#))

Number of records.

Number of container records to be removed. If the *cNumRecord* parameter has a value of 0, all of the records in the container are removed and the *pRecordArray* parameter is ignored.

CM_REMOVERECORD Field - fRemoveRecord

fRemoveRecord ([USHORT](#))

Flags.

Flags that show whether memory must be freed and container records invalidated.

CMA_FREE

If specified, [RECORDCORE](#) structures are removed and memory associated with the [RECORDCORE](#) structures is freed. If not specified, [RECORDCORE](#) structures are removed and no memory is freed; this is the default.

CMA_INVALIDATE

If specified, after [RECORDCORE](#) structures are removed the container is invalidated and any necessary repositioning of the container records is performed. If not specified, invalidation is not performed.

This option is not valid in the icon view unless the CCS_AUTOPOSITION style bit is not set. In the icon view, the container record is refreshed if the CCS_AUTOPOSITION style bit is set, regardless of whether the CMA_INVALIDATE attribute is set.

CM_REMOVERECORD Return Value - cRecords

cRecords ([LONG](#))

Number of structures.

-1

An error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_INVALID_PARAMETERS

- PMERR_MEMORY_DEALLOCATION_ERR.

Other

Number of root level [RECORDCORE](#) structures that remain in the container.

CM_REMOVERECORD - Parameters

pRecordArray ([PVOID](#))

Pointer to an array of pointers to [RECORDCORE](#) structures that are to be removed.

cNumRecord ([USHORT](#))

Number of records.

Number of container records to be removed. If the *cNumRecord* parameter has a value of 0, all of the records in the container are removed and the *pRecordArray* parameter is ignored.

fRemoveRecord ([USHORT](#))

Flags.

Flags that show whether memory must be freed and container records invalidated.

CMA_FREE

If specified, [RECORDCORE](#) structures are removed and memory associated with the [RECORDCORE](#) structures is freed. If not specified, [RECORDCORE](#) structures are removed and no memory is freed; this is the default.

CMA_INVALIDATE

If specified, after [RECORDCORE](#) structures are removed the container is invalidated and any necessary repositioning of the container records is performed. If not specified, invalidation is not performed.

This option is not valid in the icon view unless the CCS_AUTOPOSITION style bit is not set. In the icon view, the container record is refreshed if the CCS_AUTOPOSITION style bit is set, regardless of whether the CMA_INVALIDATE attribute is set.

cRecords ([LONG](#))

Number of structures.

-1

An error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_MEMORY_DEALLOCATION_ERR.

Other

Number of root level [RECORDCORE](#) structures that remain in the container.

CM_REMOVERECORD - Syntax

This message removes one, multiple, or all [RECORDCORE](#) structures from the container control.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

param1

[PVOID](#) [pRecordArray](#) /* Pointer to an array of pointers to [RECORDCORE](#) structures that are to be removed

```
param2
    USHORT  cNumRecord    /* Number of records. */
    USHORT  fRemoveRecord /* Flags. */
```

CM_REMOVERECORD - Remarks

When parent item records are removed, all associated child item records are removed, as well.

If the CMA_FREE attribute is not specified, the container control removes the specified [RECORDCORE](#) structures without freeing the memory. The application is responsible for freeing the memory associated with the [RECORDCORE](#) structure by using the [CM_FREERECORD](#) message.

If the *cNumRecord* parameter has a value of 0 and the CMA_FREE attribute is specified, all of the [RECORDCORE](#) structures in the container control are removed and the memory associated with the [RECORDCORE](#) structures is freed. It is the application's responsibility to free all of the application-allocated memory associated with the [RECORDCORE](#) structures.

If the number of pointers to [RECORDCORE](#) structures in the array exceeds the count of [RECORDCORE](#) structures to be removed, only the number of records specified in the *cNumRecord* parameter is removed. If the CCS_VERIFYPOINTERS style bit is set and the *pRecordArray* parameter contains pointers to a [RECORDCORE](#) structure or structures that do not exist, the PMERR_INVALID_PARAMETERS error is set.

If the CMA_INVALIDATE attribute is specified, the container is repainted if the removed record or records are visible.

CM_REMOVERECORD - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

CM_REMOVERECORD - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CM_SCROLLWINDOW

CM_SCROLLWINDOW Field - fsScrollDirection

fsScrollDirection (USHORT)

Scroll direction.

Direction in which to scroll the container window.

CMA_VERTICAL

Scroll vertically.

CMA_HORIZONTAL

Scroll horizontally.

CM_SCROLLWINDOW Field - IScrollInc

IScrollInc (LONG)

Scroll increment.

Amount (in pixels) by which to scroll the window.

CM_SCROLLWINDOW Return Value - rc

rc (BOOL)

Success indicator.

TRUE

Successful completion

FALSE

An error occurred. The [WinGetLastError](#) function may return the following error:
PMERR_INVALID_PARAMETERS.

CM_SCROLLWINDOW - Parameters

fsScrollDirection (USHORT)

Scroll direction.

Direction in which to scroll the container window.

CMA_VERTICAL

Scroll vertically.

CMA_HORIZONTAL

Scroll horizontally.

IScrollInc (LONG)

Scroll increment.

Amount (in pixels) by which to scroll the window.

rc (BOOL)

Success indicator.

TRUE

Successful completion

FALSE

An error occurred. The [WinGetLastError](#) function may return the following error:
PMERR_INVALID_PARAMETERS.

CM_SCROLLWINDOW - Syntax

This message scrolls an entire container window.

```
param1
    USHORT    fsScrollDirection /* Scroll direction. */

param2
    LONG      lScrollInc        /* Scroll increment. */
```

CM_SCROLLWINDOW - Remarks

If the */ScrollInc* parameter value is greater than 0 and the CMA_HORIZONTAL attribute is specified, the container window is scrolled to the right. The container window is scrolled down if the */ScrollInc* parameter value is greater than 0 and the CMA_VERTICAL attribute is specified. Similarly, the container window is scrolled left and up, respectively, if the */ScrollInc* parameter value is less than 0 and the same two attributes are specified.

If you want the container window to be scrolled by an amount that is indicated with a key, such as the PgUp, PgDn, Home, and End keys, the application can send a key event to the scroll bar.

If the container window is displaying the split details view, the [CM_HORZSCROLLSPLITWINDOW](#) message is used for horizontal scrolling.

CM_SCROLLWINDOW - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_SCROLLWINDOW - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

CM_SEARCHSTRING

CM_SEARCHSTRING Field - pSearchString

pSearchString ([PSEARCHSTRING](#))
Pointer to the [SEARCHSTRING](#) structure.

CM_SEARCHSTRING Field - pSearchAfter

pSearchAfter ([PRECORDCORE](#))
Pointer to the starting container record.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

CMA_FIRST
Start the search at the first container record.

Other
Start the search after the container record specified by this pointer. To get all of the records in the container whose text matches the string, this message is sent repeatedly. Each time this message is sent, the *pSearchAfter* parameter contains a pointer to the last record that was found.

CM_SEARCHSTRING Return Value - pRecord

pRecord ([PRECORDCORE](#))
Pointer to the found container record.

NULL
No container record's text matches the search string.

-1
An error occurred. The [WinGetLastError](#) function may return the following error:
PMERR_INVALID_PARAMETERS.

Other
Pointer to the container record whose text matches the search string.

CM_SEARCHSTRING - Parameters

pSearchString ([PSEARCHSTRING](#))

Pointer to the [SEARCHSTRING](#) structure.

pSearchAfter ([PRECORDCORE](#))

Pointer to the starting container record.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

CMA_FIRST

Start the search at the first container record.

Other

Start the search after the container record specified by this pointer. To get all of the records in the container whose text matches the string, this message is sent repeatedly. Each time this message is sent, the *pSearchAfter* parameter contains a pointer to the last record that was found.

pRecord ([PRECORDCORE](#))

Pointer to the found container record.

NULL

No container record's text matches the search string.

-1

An error occurred. The [WinGetLastError](#) function may return the following error:

PMERR_INVALID_PARAMETERS.

Other

Pointer to the container record whose text matches the search string.

CM_SEARCHSTRING - Syntax

This message returns the pointer to a container record whose text matches the string.

```
param1
    PSEARCHSTRING  pSearchString /* Pointer to the SEARCHSTRING structure. */

param2
    PRECORDCORE    pSearchAfter  /* Pointer to the starting container record. */
```

CM_SEARCHSTRING - Remarks

The CM_SEARCHSTRING message is NLS-enabled.

In the details view, the string is searched for in each column of each record.

CM_SEARCHSTRING - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return NULL.

CM_SEARCHSTRING - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

CM_SETCNRINFO

CM_SETCNRINFO Field - pCnrInfo

pCnrInfo ([PCNRINFO](#))
Pointer to the [CNRINFO](#) structure from which to set the data for the container.

CM_SETCNRINFO Field - ulCnrInfoFI

- ulCnrInfoFI** ([ULONG](#))
Flags.
- Flags that show which fields are to be set.
- CMA_PSORTRECORD**
Pointer to the comparison function for sorting container records. If NULL, which is the default condition, no sorting is performed. Sorting only occurs during record insertion and when changing the value of this field. The third parameter of the comparison function, *pStorage*, must be NULL. See [CM_SORTRECORD](#) for a further description of the comparison function.
- CMA_PFIELDINFOLAST**
Pointer to the last column in the left window of the split details view. The default is NULL, causing all columns to be positioned in the left window.
- CMA_PFIELDINFOBJECT**
Pointer to a column that represents an object in the details view. This [FIELDINFO](#) structure must contain icons or bit maps. In-use emphasis is applied to this column of icons or bit maps only. The default is the leftmost column in the unsplit details view, or the leftmost column in the left window of the split details view.
- CMA_CNRTITLE**
Text for the container title. The default is NULL.
- CMA_FLWINDOWATTR**
Container window attributes.
- CMA_PTLORIGIN**

Lower-left origin of the container window in virtual workspace coordinates, used in the icon view. The default origin is **(0,0)**.

CMA_DELTA

An application-defined threshold, or number of records, from either end of the list of available records. Used when a container needs to handle large amounts of data. The default is 0. Refer to the description of the container control in the *OS/2 Programming Guide* for more information about specifying deltas.

CMA_SLBITMAPORICON

The size (in pels) of icons or bit maps. The default is the system size.

CMA_SLTREEBITMAPORICON

The size (in pels) of the expanded and collapsed icons or bit maps in the tree icon and tree text views.

CMA_TREEBITMAP

Expanded and collapsed bit maps in the tree icon and tree text views.

CMA_TREEICON

Expanded and collapsed icons in the tree icon and tree text views.

CMA_LINESPACING

The amount of vertical space (in pels) between the records. If this value is less than 0, a default value is used.

CMA_CXTREEINDENT

Horizontal distance (in pels) between levels in the tree view. If this value is less than 0, a default value is used.

CMA_CXTREELINE

Width of the lines (in pels) that show the relationship between items in the tree view. If this value is less than 0, a default value is used. Also, if the CA_TREELINE container attribute of the [CNRINFO](#) data structure's *flWindowAttr* field is not specified, these lines are not drawn.

CMA_XVERTSPLITBAR

The initial position of the split bar relative to the container, used in the details view. If this value is less than 0, the split bar is not used. The default value is negative one (-1).

CM_SETCNRINFO Parameter - rc

rc ([BOOL](#))

Success indicator.

TRUE

Container data was successfully set.

FALSE

Container data was not set. The [WinGetLastError](#) function may return the following errors:

- [PMERR_INVALID_PARAMETERS](#)
- [PMERR_INSUFFICIENT_MEMORY](#).

CM_SETCNRINFO - Parameters

pCnrInfo ([PCNRINFO](#))

Pointer to the [CNRINFO](#) structure from which to set the data for the container.

ulCnrInfoFI ([ULONG](#))

Flags.

Flags that show which fields are to be set.

CMA_PSORTRECORD

Pointer to the comparison function for sorting container records. If NULL, which is the default condition, no sorting is performed. Sorting only occurs during record insertion and when changing the value of this field. The third parameter of the comparison function, *pStorage*, must be NULL. See [CM_SORTRECORD](#) for a further description of the comparison function.

CMA_PFIELDINFOLAST

Pointer to the last column in the left window of the split details view. The default is NULL, causing all columns to be positioned in the left window.

CMA_PFIELDINFOBJECT

Pointer to a column that represents an object in the details view. This [FIELDINFO](#) structure must contain icons or bit maps. In-use emphasis is applied to this column of icons or bit maps only. The default is the leftmost column in the unsplit details view, or the leftmost column in the left window of the split details view.

CMA_CNRTITLE

Text for the container title. The default is NULL.

CMA_FLWINDOWATTR

Container window attributes.

CMA_PTLORIGIN

Lower-left origin of the container window in virtual workspace coordinates, used in the icon view. The default origin is **(0,0)**.

CMA_DELTA

An application-defined threshold, or number of records, from either end of the list of available records. Used when a container needs to handle large amounts of data. The default is 0. Refer to the description of the container control in the *OS/2 Programming Guide* for more information about specifying deltas.

CMA_SLBITMAPORICON

The size (in pels) of icons or bit maps. The default is the system size.

CMA_SLTREEBITMAPORICON

The size (in pels) of the expanded and collapsed icons or bit maps in the tree icon and tree text views.

CMA_TREEBITMAP

Expanded and collapsed bit maps in the tree icon and tree text views.

CMA_TREEICON

Expanded and collapsed icons in the tree icon and tree text views.

CMA_LINESPACING

The amount of vertical space (in pels) between the records. If this value is less than 0, a default value is used.

CMA_CXTREEINDENT

Horizontal distance (in pels) between levels in the tree view. If this value is less than 0, a default value is used.

CMA_CXTREELINE

Width of the lines (in pels) that show the relationship between items in the tree view. If this value is less than 0, a default value is used. Also, if the CA_TREELINE container attribute of the [CNRINFO](#) data structure's *fWindowAttr* field is not specified, these lines are not drawn.

CMA_XVERTSPLITBAR

The initial position of the split bar relative to the container, used in the details view. If this value is less than 0, the split bar is not used. The default value is negative one (-1).

rc ([BOOL](#))

Success indicator.

TRUE

Container data was successfully set.

FALSE

Container data was not set. The [WinGetLastError](#) function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_INSUFFICIENT_MEMORY.

CM_SETCNRINFO - Syntax

This message sets or changes the data for the container control.

```
param1
    PCNRINFO  pCnrInfo      /*  Pointer to the CNRINFO structure from which to set the data for the container.

param2
    ULONG     ulCnrInfoFl   /*  Flags.  */
```

CM_SETCNRINFO - Remarks

The data for a container is set from the buffer addressed by the *pCnrInfo* parameter. The flags in the *ulCnrInfoFl* parameter show which part or parts of the *pCnrInfo* parameter are set. The flag values can be combined by using a logical OR operator (|).

CM_SETCNRINFO - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_SETCNRINFO - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

CM_SETGRIDINFO

CM_SETGRIDINFO Field - PGRIDINFO

PGRIDINFO ([PGRIDINFO](#))
Pointer to a GRIDINFO structure.

CM_SETGRIDINFO Field - bRepaint

bRepaint (BOOL)

Repaint the window at this time?

TRUE

If this field is TRUE, the container is immediately invalidated and repainted.

FALSE

If this field is FALSE, no repositioning or repainting of items occurs. If the application sets this field to FALSE, it must send a [CM_ARRANGE](#) or [CM_SNAPTOGRID](#) message afterwards.

CM_SETGRIDINFO Return Value - rc

rc (BOOL)

Success indicator.

TRUE

Grid characteristics were set successfully.

FALSE

An error occurred while setting grid information; grid has not been changed.

CM_SETGRIDINFO - Parameters

PGRIDINFO (PGRIDINFO)

Pointer to a GRIDINFO structure.

bRepaint (BOOL)

Repaint the window at this time?

TRUE

If this field is TRUE, the container is immediately invalidated and repainted.

FALSE

If this field is FALSE, no repositioning or repainting of items occurs. If the application sets this field to FALSE, it must send a [CM_ARRANGE](#) or [CM_SNAPTOGRID](#) message afterwards.

rc (BOOL)

Success indicator.

TRUE

Grid characteristics were set successfully.

FALSE

An error occurred while setting grid information; grid has not been changed.

CM_SETGRIDINFO - Syntax

This message sets the characteristics of the grid.

```
param1
    PGRIDINFO PGRIDINFO /* Pointer to a GRIDINFO structure. */

param2
    BOOL      bRepaint   /* Repaint the window at this time? */
```

CM_SETGRIDINFO - Remarks

This message can be used to change the size of the grid squares after a grid has been created.

The only fields in the GRIDINFO structure that can be set by the application are the *cb*, *cxGrid*, *cyGrid*, and *pGrid* fields. If the *cxGrid* and *cyGrid* fields are nonzero and the *pGrid* field is NULL, the container creates a grid with grid squares of width *cxGrid* and height *cyGrid*.

If *cxGrid* and *cyGrid* are zero, the container will create default size grid squares. It is not necessary to send this message to create default size grid squares, because they will be created the first time an arrange with the CM_ARRANGEGRID option is performed.

The application can completely control the placement of icons by marking the squares it wants occupied and then sending the CM_ARRANGE message with *mp1* equal to CMA_ARRANGEGRID and *mp2* equal to CMA_USER. Here is how the application uses CM_SETGRIDINFO to mark the squares:

- If the *pGrid* field of the GRIDINFO structure is non-NULL, the container expects a pointer that points to an array of GRIDSQUARE structures, one for each available grid square in the grid. In this case, the *cxGrid* and *cyGrid* fields are ignored.
- The container uses the grid square array to mark the grid squares as available or unavailable. If the *uiState* field of the GRIDSQUARE structure is marked CMA_AVAIL, the square can be occupied by an icon; if it is marked CMA_UNAVAIL, it is not available.

Marking of grid squares is used only when the arrangement pattern is CMA_USER and has no effect on the placement of icons when any other pattern is specified.

CM_SETGRIDINFO - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_SETGRIDINFO - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CM_SETRECORDEMPHASIS

CM_SETRECORDEMPHASIS Field - pRecord

pRecord ([PRECORDCORE](#))
Pointer to the specified container record.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

CM_SETRECORDEMPHASIS Field - usChangeEmphasis

usChangeEmphasis ([USHORT](#))
Change-emphasis-attribute flag.

- TRUE
- The container record's emphasis attribute is to be set ON if the change specified is not the same as the current state.
- FALSE
- The container record's emphasis attribute is to be set OFF if the change specified is not the same as the current state.
-

CM_SETRECORDEMPHASIS Field - fEmphasisAttribute

fEmphasisAttribute ([USHORT](#))
Emphasis attribute of the container record.

The following states can be combined by using a logical OR operator (|):

- CRA_CURSORED
- Specifies that a record will be drawn with a selection cursor.
- CRA_DISABLED
- Specifies that a record will be drawn with unavailable-state emphasis.
- CRA_INUSE
- Specifies that a record will be drawn with in-use emphasis.
- CRA_PICKED
- Specifies that the container record will be picked up as part of the drag set.
- CRA_SELECTED
- Specifies that a record will be drawn with selected-state emphasis.
- CRA_SOURCE
- Specifies that a record will be drawn with source-menu emphasis.
-

CM_SETRECORDEMPHASIS Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	An error occurred.

The [WinGetLastError](#) function may return the following errors:

PMERR_INVALID_PARAMETERS (1208)
PMERR_INSUFFICIENT_MEMORY (203E)

CM_SETRECORDEMPHASIS - Parameters

pRecord (PRECORDCORE)
Pointer to the specified container record.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

usChangeEmphasis (USHORT)
Change-emphasis-attribute flag.

TRUE	The container record's emphasis attribute is to be set ON if the change specified is not the same as the current state.
FALSE	The container record's emphasis attribute is to be set OFF if the change specified is not the same as the current state.

fEmphasisAttribute (USHORT)
Emphasis attribute of the container record.

The following states can be combined by using a logical OR operator (|):

CRA_CURSORED	Specifies that a record will be drawn with a selection cursor.
CRA_DISABLED	Specifies that a record will be drawn with unavailable-state emphasis.
CRA_INUSE	Specifies that a record will be drawn with in-use emphasis.
CRA_PICKED	Specifies that the container record will be picked up as part of the drag set.
CRA_SELECTED	Specifies that a record will be drawn with selected-state emphasis.
CRA_SOURCE	Specifies that a record will be drawn with source-menu emphasis.

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	An error occurred.

The [WinGetLastError](#) function may return the following errors:

PMERR_INVALID_PARAMETERS (1208)

CM_SETRECORDEMPHASIS - Syntax

This message sets the emphasis attributes of the specified container record.

```
param1
    RECORDCORE    pRecord          /* Pointer to the specified container record. */

param2
    USHORT        usChangeEmphasis /* Change-emphasis-attribute flag. */
    USHORT        fEmphasisAttribute /* Emphasis attribute of the container record. */
```

CM_SETRECORDEMPHASIS - Remarks

For single-selection containers, the selection of the previous container record is cancelled before another record is selected. The selection cursor is set with the CRA_CURSORED attribute for single-selection containers. Only one selection cursor is allowed.

The selection cursor must always be available to the user. Therefore, if you attempt to disable the selection cursor by specifying FALSE for the *usChangeEmphasis* parameter and CRA_CURSORED for the *fEmphasisAttribute* parameter, the PMERR_INVALID_PARAMETERS error is set. In order to change the selection cursor attribute, TRUE should be specified for the *usChangeEmphasis* parameter and CRA_CURSORED for the *fEmphasisAttribute* parameter. The *pRecord* parameter should point to the record to which the selection cursor should be applied. The container control removes the selection cursor from the record with the cursor and applies it to the new record.

A [CN_EMPHASIS](#) notification code is sent to the container owner if the record emphasis attribute is changed.

CM_SETRECORDEMPHASIS - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_SETRECORDEMPHASIS - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

CM_SETTEXTVISIBILITY

CM_SETTEXTVISIBILITY Field - bVisible

bVisible (BOOL)
Text visibility state.

TRUE	Text is visible.
FALSE	Text is not visible.

CM_SETTEXTVISIBILITY Field - Reserved

Reserved (PVOID)
Reserved value, 0.

CM_SETTEXTVISIBILITY Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Text visibility state was successfully set.
FALSE	Error occurred.

CM_SETTEXTVISIBILITY - Parameters

bVisible (BOOL)
Text visibility state.

TRUE	Text is visible.
FALSE	Text is not visible.

Reserved (PVOID)
Reserved value, 0.

rc (BOOL)
Success indicator.

TRUE	Text visibility state was successfully set.
FALSE	Error occurred.

CM_SETTEXTVISIBILITY - Syntax

This message sets the visibility state of text for records in the container control.

```
param1
    BOOL    bVisible /* Text visibility state. */

param2
    PVOID    Reserved /* Reserved value, 0. */
```

CM_SETTEXTVISIBILITY - Remarks

This message is used to set the visibility state of the text for records in the container control. If *bVisible* is TRUE, text will appear with the icons in icon view, name view, tree icon view, and tree name view. If *bVisible* is FALSE, no text appears.

This message does not apply to any variation of text view (icon text, tree text) or details view.

This message affects ALL records within the container. The visibility state of the text cannot be set for individual records.

CM_SETTEXTVISIBILITY - Default Processing

The default window procedure does not expect to receive this message and, therefore, takes no action on it other than to return FALSE.

CM_SETTEXTVISIBILITY - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

CM_SNAPTOGRID

CM_SNAPTOGRID Field - preccList

preccList ([PRECORDCORE](#))
Pointer to a linked list of container records to be repositioned in the grid.

CM_SNAPTOGRID Field - xDrop

xDrop ([SHORT](#))
X-coordinate of drop point expressed in desktop coordinates.

CM_SNAPTOGRID Field - yDrop

yDrop ([SHORT](#))
Y-coordinate of drop point expressed in desktop coordinates.

CM_SNAPTOGRID Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Records were successfully snapped to the grid.
FALSE	An error occurred.

CM_SNAPTOGRID - Parameters

preccList ([PRECORDCORE](#))
Pointer to a linked list of container records to be repositioned in the grid.

xDrop ([SHORT](#))
X-coordinate of drop point expressed in desktop coordinates.

yDrop ([SHORT](#))
Y-coordinate of drop point expressed in desktop coordinates.

rc ([BOOL](#))

Success indicator.

TRUE

Records were successfully snapped to the grid.

FALSE

An error occurred.

CM_SNAPTOGRID - Syntax

This message supports the snap-to-grid property, which aligns icons within grid squares whenever they are dragged and dropped inside the container.

```
param1
    RECORDCORE precclist /* Pointer to a linked list of container records to be repositioned in the grid.

param2
    SHORT      xDrop      /* X-coordinate of drop point expressed in desktop coordinates. */
    SHORT      yDrop      /* Y-coordinate of drop point expressed in desktop coordinates. */
```

CM_SNAPTOGRID - Remarks

Because of the implementation of direct manipulation in OS/2, the snap-to-grid property requires cooperation from the application.

Items being dragged are represented by DRAGITEM structures. This structure contains the *ulItemID* field, which is available for application use. Applications usually use this field to identify the object being dragged. To provide flexibility to applications, the value in the field can vary from application to application. The most natural value for this field when dragging objects within a container would be the RECORDCORE pointer. However the field can also be a pointer to an application-defined structure that contains many fields (possibly including the RECORDCORE pointer).

Because *ulItemID* can have more than one interpretation, the container must be told which records are to be repositioned and where to begin the repositioning. An application can do this by iterating through the DRAGITEM structures (provided by the CN_DROP notification) and creating a linked list of container records to be repositioned or snapped to the grid. The point to begin the positioning from can be obtained directly from the DRAGINFO structure passed on the CN_DROP notification.

Grid squares that are marked partial (the CMA_PARTIAL bit is set in the *ulState* field of the GRIDSQUARE structure) can be used for the Snap-To-Grid property.

Note: This message returns FALSE if the current view is not CV_GRID.

CM_SNAPTOGRID - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_SNAPTOGRID - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

CM_SORTRECORD

CM_SORTRECORD Field - pfnCompare

pfnCompare ([PFN](#))
Pointer to a comparison function.

CM_SORTRECORD Field - pStorage

pStorage ([PVOID](#))
Application use.

Available for application use.

CM_SORTRECORD Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE
The records in the container were sorted.

FALSE
The records in the container were not sorted. The [WinGetLastError](#) function may return the following errors:

- PMERR_COMPARISON_FAILED
- PMERR_INSUFFICIENT_MEMORY.

CM_SORTRECORD - Parameters

pfnCompare (PFN)
Pointer to a comparison function.

pStorage (PVOID)
Application use.

Available for application use.

rc (BOOL)
Success indicator.

TRUE
The records in the container were sorted.

FALSE
The records in the container were not sorted. The [WinGetLastError](#) function may return the following errors:

- PMERR_COMPARISON_FAILED
- PMERR_INSUFFICIENT_MEMORY.

CM_SORTRECORD - Syntax

This message sorts the container records in the container control.

```
param1
    PFN    pfnCompare /* Pointer to a comparison function. */

param2
    PVOID  pStorage   /* Application use. */
```

CM_SORTRECORD - Remarks

The *pfnCompare* parameter must be declared as:

```
SHORT EXPENTRY pfnCompare(
    PRECORDCORE p1,
    PRECORDCORE p2,
    PVOID pStorage);
```

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

The *pfnCompare* parameter points to an application-provided function that compares two [RECORDCORE](#) structures and returns a [SHORT](#) value that specifies their relationship. The *pfnCompare* parameter is called one or more times during the sorting process and is passed pointers to two [RECORDCORE](#) structures on each call. The routine must compare the [RECORDCORE](#) structures, and then return one of the following values:

<u>Value</u>	<u>Meaning</u>
>0	<i>p1</i> is less than <i>p2</i> .
0	<i>p1</i> is equal to <i>p2</i> .
<0	<i>p1</i> is greater than <i>p2</i> .

The container records are sorted in increasing order, as defined by the *pfnCompare* parameter. The records can be sorted in reverse order by reversing the sense of "greater than" and "less than" in the *pfnCompare* parameter.

If the container has only one record, the PMERR_COMPARISON_FAILED error is set.

The application must provide an NLS-enabled function for the *pfnCompare* parameter. The container control does not provide NLS enablement for sorting.

An alternative to using the CM_SORTRECORD message is to provide an application-defined comparison function to sort the container records, which can be specified in the CNRINFO structure's *pSortRecord* field. If this function is provided, the container records are sorted as they are inserted into the container control. If this field is NULL, the records are not sorted on insertion.

CM_SORTRECORD - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

CM_SORTRECORD - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_PICKUP

WM_PICKUP Field - ptlPointerPos

ptlPointerPos ([POINTL](#))
Pointer position in window coordinates relative to the bottom-left corner of the window.

WM_PICKUP Field - Reserved

Reserved ([ULONG](#))
Reserved value, must be 0.

WM_PICKUP Field - rc

rc (BOOL)
Success indicator.

Possible values are described in the following list:

TRUE	Message was processed.
FALSE	Message was ignored.

WM_PICKUP - Parameters

ptlPointerPos (POINTL)
Pointer position in window coordinates relative to the bottom-left corner of the window.

Reserved (ULONG)
Reserved value, must be 0.

rc (BOOL)
Success indicator.

Possible values are described in the following list:

TRUE	Message was processed.
FALSE	Message was ignored.

WM_PICKUP - Syntax

This message adds objects to the drag set during a lazy drag operation.

```
param1
    POINTL  ptlPointerPos /* Pointer position in window coordinates relative to the bottom-left corner of th

param2
    ULONG   Reserved      /* Reserved value, must be 0. */

returns
    BOOL    rc            /* Success indicator. */
```

WM_PICKUP - Remarks

This message will be posted to the application queue associated with the window that has the focus, or with the window that is to receive the pointer-button information.

WM_PICKUP message is sent to the window under the mouse pointer when the user presses the direct-manipulation button while holding down the lazy drag augmentation key, currently the ALT key. This message is used to inform an application that the user is commencing a lazy drag operation. The container control sends its owner a [CN_PICKUP](#) notification when it receives a message.

Objects are added to the drag set when a WM_PICKUP message is received. The first time the message is received, the application initiates a lazy drag operation. Each subsequent WM_PICKUP message that is received during the course of the lazy drag operation indicates that objects are to be added to the drag set.

WM_PICKUP - Default Processing

The default message procedure sets *rc* to TRUE.

WM_PICKUP - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_PRESPARAMCHANGED (in Container Controls)

WM_PRESPARAMCHANGED (in Container Controls) Field - attrtype

attrtype ([ULONG](#))

Presentation parameter attribute identity.

PP_BACKGROUND_COLOR or PP_BACKGROUND_COLOR_INDEX

Sets the background color of the container window. This color is initially set to SYSCLR_WINDOW.

PP_BORDER_COLOR or PP_BORDER_COLOR_INDEX

Sets the color of the title separators, column separators, and split bar. This color is initially set to SYSCLR_WINDOWFRAME.

PP_FONT_NAME_SIZE

Sets the font and font size of the text in the container. This font and font size defaults to the system font and font size.

PP_FOREGROUND_COLOR or PP_FOREGROUND_COLOR_INDEX

Sets the color of unselected text. This color is initially set to SYSCLR_WINDOWTEXT.

PP_HILITEBACKGROUNDCOLOR or PP_HILITEBACKGROUNDINDEX

Sets the color of selection emphasis, the color of the cursor of an unselected item in the details view, and the color of the cursor in all other views. This color is initially set to SYSCLR_HILITEBACKGROUND.

PP_HILITEFOREGROUNDINDEX or PP_HILITEFOREGROUNDINDEX

Sets the color of the text of a selected item in all views and the color of the cursor of a selected item in the details view. This color is initially set to SYSCLR_HILITEFOREGROUND.

WM_PRESPARAMCHANGED (in Container Controls) Field - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

WM_PRESPARAMCHANGED (in Container Controls) Return Value - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

WM_PRESPARAMCHANGED (in Container Controls) - Parameters

attrtype (ULONG)

Presentation parameter attribute identity.

PP_BACKGROUNDINDEX or PP_BACKGROUNDINDEX

Sets the background color of the container window. This color is initially set to SYSCLR_WINDOW.

PP_BORDERINDEX or PP_BORDERINDEX

Sets the color of the title separators, column separators, and split bar. This color is initially set to SYSCLR_WINDOWFRAME.

PP_FONTNAMEINDEX

Sets the font and font size of the text in the container. This font and font size defaults to the system font and font size.

PP_FOREGROUNDINDEX or PP_FOREGROUNDINDEX

Sets the color of unselected text. This color is initially set to SYSCLR_WINDOWTEXT.

PP_HILITEBACKGROUNDINDEX or PP_HILITEBACKGROUNDINDEX

Sets the color of selection emphasis, the color of the cursor of an unselected item in the details view, and the color of the cursor in all other views. This color is initially set to SYSCLR_HILITEBACKGROUND.

PP_HILITEFOREGROUNDINDEX or PP_HILITEFOREGROUNDINDEX

Sets the color of the text of a selected item in all views and the color of the cursor of a selected item in the details view. This color is initially set to `SYSCLR_HILITEFOREGROUND`.

ulReserved ([ULONG](#))

Reserved value, should be 0.

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_PRESPARAMCHANGED (in Container Controls) - Syntax

For the cause of this message, see [WM_PRESPARAMCHANGED](#).

```
param1
    ULONG attrtype    /* Presentation parameter attribute identity. */

param2
    ULONG ulReserved  /* Reserved value, should be 0. */
```

WM_PRESPARAMCHANGED (in Container Controls) - Remarks

The application uses the [WinSetPresParam](#) function to change presentation parameters. This results in a `WM_PRESPARAMCHANGED` (in Container Controls) message being sent to the container.

WM_PRESPARAMCHANGED (in Container Controls) - Default Processing

For a description of the default processing, see [WM_PRESPARAMCHANGED](#).

WM_PRESPARAMCHANGED (in Container Controls) - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

Default Dialog Processing

This section describes how messages are processed by the default dialog procedure. The default dialog procedure can be called using [WinDefDlgProc](#). A user dialog procedure should make this call for all messages that it does not want to process.

For WM_* messages other than those specified in this section the Default Dialog Procedure takes the same action and sets **result** to the same value as in [Frame Control Window Processing](#). In the instance of messages that would be sent to FID_CLIENT, they are passed to the default window procedure.

For any other messages the default window procedure takes no action, other than to set **reply** to NULL.

Default Dialog Messages

This section describes the default dialog procedure actions on receiving the following messages.

WM_CHAR (Default Dialogs)

WM_CHAR (Default Dialogs) - Syntax

For the cause of this message, see [WM_CHAR](#).

For a description of the parameters, see [WM_CHAR](#).

WM_CHAR (Default Dialogs) - Default Processing

If KC_CHAR is the mnemonic for a button that already has the focus, a [BM_CLICK](#) is sent to that button and *rc* is set to TRUE. If the button does not have the focus, it receives the focus and *rc* is set to TRUE.

If *usvk* contains the value VK_TAB, the focus is set to the next tab item in the dialog. *rc* is set to TRUE.

If *usvk* contains the value VK_BACKTAB, the focus is set to the previous tab item in the dialog. *rc* is set to TRUE.

If *usvk* contains the value VK_LEFT or VK_UP, the focus is set to the previous item in the group. *rc* is set to TRUE.

If *usvk* contains the value VK_RIGHT or VK_BOTTOM, the focus is set to the next item in the group. *rc* is set to TRUE.

If *usvk* contains the value VK_ENTER or VK_NEWLINE, and a push button has the focus, a [BM_CLICK](#) is sent to the button and *rc* is set to TRUE. If another control in the dialog has the focus the dialog is searched for a push button with style BS_DEFAULT. If a push button of this style is found, a [BM_CLICK](#) is sent to that button and *rc* is set to TRUE.

If *usvk* contains the value VK_ESC, [WM_COMMAND](#) is posted, with *ussource* is set to CMDSRC_PUSHBUTTON and *uscmd* is set to

DID_CANCEL. *rc* is set to TRUE.

In other instances, if an owner exists the message is sent to the owner, otherwise *rc* is set to FALSE.

WM_CHAR (Default Dialogs) - Related Messages

Related Messages

- [WM_CHAR](#)
-

WM_CHAR (Default Dialogs) - Topics

Select an item:

[Syntax](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_CLOSE (Default Dialogs)

WM_CLOSE (Default Dialogs) - Syntax

For the cause of this message, see [WM_CLOSE](#).

For a description of the parameters, see [WM_CLOSE](#).

WM_CLOSE (Default Dialogs) - Default Processing

The default dialog procedure responds to this message by dismissing the dialog by issuing the [WinDismissDlg](#) function with its parameter set to DID_CANCEL.

WM_CLOSE (Default Dialogs) - Related Messages

Related Messages

- [WM_CLOSE](#)

WM_CLOSE (Default Dialogs) - Topics

Select an item:

[Syntax](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_COMMAND (Default Dialogs)

WM_COMMAND (Default Dialogs) - Syntax

For the cause of this message, see [WM_COMMAND](#).

For a description of the parameters, see [WM_COMMAND](#).

WM_COMMAND (Default Dialogs) - Default Processing

The default dialog procedure responds to this message by dismissing the dialog and passing *uscmd* (the control item identifier) as of the [WinProcessDlg](#) or the [WinDlgBox](#) function that initiated the dialog. It sets *ulReserved* to 0.

WM_COMMAND (Default Dialogs) - Related Messages

Related Messages

- [WM_COMMAND](#)
-

WM_COMMAND (Default Dialogs) - Topics

Select an item:

[Syntax](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_INITDLG (Default Dialogs)

WM_INITDLG (Default Dialogs) - Syntax

For the cause of this message, see [WM_INITDLG](#).

For a description of the parameters, see [WM_INITDLG](#).

WM_INITDLG (Default Dialogs) - Remarks

This message is sent to the dialog procedure, before the dialog box is shown, thereby offering the dialog procedure the opportunity to perform the initialization of the dialog box.

If any string substitutions are made by the [WinSubstituteStrings](#) call when the dialog is created, the [WM_SUBSTITUTESTRING](#) message may have been sent before the [WM_INITDLG](#) message is sent.

WM_INITDLG (Default Dialogs) - Default Processing

The default dialog procedure passes this message to the default window procedure, which sets *rc* to FALSE.

WM_INITDLG (Default Dialogs) - Related Messages

Related Messages

- [WM_INITDLG](#)
-

WM_INITDLG (Default Dialogs) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_MATCHMNEMONIC (Default Dialogs)

WM_MATCHMNEMONIC (Default Dialogs) - Syntax

For the cause of this message, see [WM_MATCHMNEMONIC](#).

For a description of the parameters, see [WM_MATCHMNEMONIC](#).

WM_MATCHMNEMONIC (Default Dialogs) - Remarks

This message is only processed by Button and Static Controls; all other controls return FALSE.

WM_MATCHMNEMONIC (Default Dialogs) - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_MATCHMNEMONIC (Default Dialogs) - Related Messages

Related Messages

- [WM_MATCHMNEMONIC](#)
-

WM_MATCHMNEMONIC (Default Dialogs) - Topics

Select an item:

[Syntax](#)

[Remarks](#)

[Default Processing](#)

[Related Messages](#)

[Glossary](#)

WM_QUERYDLGCODE

WM_QUERYDLGCODE Field - pQmsg

pQmsg ([PQMSG](#))

Message queue structure.

This points to a [QMSG](#) structure.

WM_QUERYDLGCODE Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_QUERYDLGCODE Return Value - ulDialogCode

ulDialogCode ([ULONG](#))

Dialog code information flags.

DLGC_ENTRYFIELD

Identifies an entry field control. Assumed to understand the [EM_SETSEL](#) message.

DLGC_BUTTON

Identifies a button item. Assumed to understand the [BM_CLICK](#) message.

DLGC_RADIOBUTTON

Identifies a radio button control. Used with the DLGC_BUTTON code.

DLGC_STATIC

Identifies a static control. Static controls are not included in arrow key enumeration.

DLGC_DEFAULT

Identifies a default push-button control.

DLGC_PUSHBUTTON

Identifies a nondefault push button.

DLGC_CHECKBOX

Identifies a check-box item. Used with the DLGC_BUTTON code.

DLGC_SCROLLBAR

Identifies a scroll bar control.

DLGC_MENU

Identifies a menu control.

DLGC_TABONCLICK

Used by static controls to indicate that a mouse click on this control will cause focus to be placed on the next control in the dialog that has the WP_TABSTOP style. This should be used in combination with the DLGC_STATIC code.

DLGC_MLE

Identifies a multiline entry field control.

WM_QUERYDLGCODE - Parameters

pQmsg (**PQMSG**)

Message queue structure.

This points to a **QMSG** structure.

ulReserved (**ULONG**)

Reserved value, should be 0.

ulDialogCode (**ULONG**)

Dialog code information flags.

DLGC_ENTRYFIELD

Identifies an entry field control. Assumed to understand the **EM_SETSEL** message.

DLGC_BUTTON

Identifies a button item. Assumed to understand the **BM_CLICK** message.

DLGC_RADIOBUTTON

Identifies a radio button control. Used with the **DLGC_BUTTON** code.

DLGC_STATIC

Identifies a static control. Static controls are not included in arrow key enumeration.

DLGC_DEFAULT

Identifies a default push-button control.

DLGC_PUSHBUTTON

Identifies a nondefault push button.

DLGC_CHECKBOX

Identifies a check-box item. Used with the **DLGC_BUTTON** code.

DLGC_SCROLLBAR

Identifies a scroll bar control.

DLGC_MENU

Identifies a menu control.

DLGC_TABONCLICK

Used by static controls to indicate that a mouse click on this control will cause focus to be placed on the next control in the dialog that has the **WP_TABSTOP** style. This should be used in combination with the **DLGC_STATIC** code.

DLGC_MLE

Identifies a multiline entry field control.

WM_QUERYDLGCODE - Syntax

This message is sent by the dialog manager to identify the type of control, to determine what kinds of messages the control understands, and also to determine whether an input message may be processed by the dialog manager or passed down to the control.

```
param1
    PQMSG  pQmsg          /* Message queue structure. */

param2
    ULONG  ulReserved     /* Reserved value, should be 0. */
```

WM_QUERYDLGCODE - Remarks

When processing user input, the dialog manager makes some assumptions about the operation of specific controls. The dialog manager sends the WM_QUERYDLGCODE message to obtain a code that governs what assumptions can be made.

If the window receiving this message is *not* a control as defined above, this message returns 0.

WM_QUERYDLGCODE - Default Processing

The default dialog procedure takes no action on this message, other than to set *ulDialogCode* to NULL.

WM_QUERYDLGCODE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

Default File Dialog Processing

This section describes how messages are processed by the default dialog procedure of the file dialog. This standard dialog can be used to provide a common, consistent file selection function.

The file dialog's default procedure can be called using the WinDefFileDlgProc function. A user-provided subclassing dialog procedure should make this call for all messages that it does not process when using the file dialog.

The default dialog procedure of the file dialog sends the messages listed in this section to itself to perform the requested action. This design allows a user-provided dialog procedure to customize the file dialog to its own needs.

File Dialog Messages

This section describes the file dialog procedure actions on receiving the following messages.

FDM_ERROR

FDM_ERROR Field - usErrorId

usErrorId ([USHORT](#))
Error message ID.

This is the ID of the message that is displayed by the file dialog if the default file dialog procedure processes the message.

FDM_ERROR Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

FDM_ERROR Return Value - usUserReply

usUserReply ([USHORT](#))
User's reply.

- | | |
|-------------|---|
| 0 | The file dialog presents the error message for this ID. |
| MBID_OK | The file dialog processes the reply as if the OK push button was pressed in its message window. |
| MBID_CANCEL | The file dialog processes the reply as if the Cancel push button was pressed in its message window. |
| MBID_RETRY | The file dialog processes the reply as if the Retry push button was pressed in its message window. |

FDM_ERROR - Parameters

usErrorId ([USHORT](#))
Error message ID.

This is the ID of the message that is displayed by the file dialog if the default file dialog procedure processes the message.

ulReserved ([ULONG](#))
Reserved value, should be 0.

usUserReply ([USHORT](#))
User's reply.

0	The file dialog presents the error message for this ID.
MBID_OK	The file dialog processes the reply as if the OK push button was pressed in its message window.
MBID_CANCEL	The file dialog processes the reply as if the Cancel push button was pressed in its message window.
MBID_RETRY	The file dialog processes the reply as if the Retry push button was pressed in its message window.

FDM_ERROR - Syntax

This message is sent whenever the file dialog is going to display an error message window. This allows an application to display its own message, if desired, instead of messages provided by the system.

```
param1
    USHORT    usErrorId    /* Error message ID. */

param2
    ULONG     ulReserved   /* Reserved value, should be 0. */
```

FDM_ERROR - Remarks

The application uses this message to provide application-specific error messages in response to file dialog errors that are detected during file dialog processing. The application can choose whether to allow the dialog to present its message or whether to provide its own message and return the response from that message window to the dialog for processing.

FDM_ERROR - Default Processing

The [WinDefDlgProc](#) function does not expect to receive this message and takes no action on it other than to return NULL.

FDM_ERROR - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

FDM_FILTER

FDM_FILTER Field - pFilename

pFilename (PSZ)
Pointer to the file name.

FDM_FILTER Field - pEAType

pEAType (PSZ)
Pointer to the .TYPE EA extended attribute.

FDM_FILTER Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Add the file.
FALSE	Do not add the file.

FDM_FILTER - Parameters

pFilename (PSZ)
Pointer to the file name.

pEAType (PSZ)
Pointer to the .TYPE EA extended attribute.

rc (BOOL)
Success indicator.

TRUE	Add the file.
FALSE	Do not add the file.

FDM_FILTER - Syntax

This message is sent before a file that meets the current filter criteria is added to the File list box.

```
param1
    PSZ    pFilename /* Pointer to the file name. */

param2
    PSZ    pEAType   /* Pointer to the .TYPE EA extended attribute. */
```

FDM_FILTER - Remarks

The application checks this message to obtain the name and the .TYPE EA extended attribute of the file to be added. The application then determines whether or not the file will be added.

When FALSE is returned, the file is not added to the dialog's list box.

FDM_FILTER - Default Processing

The [WinDefDlgProc](#) function does not expect to receive this message and takes no action on it other than to return FALSE.

FDM_FILTER - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

FDM_VALIDATE

FDM_VALIDATE Field - pFileName

pFileName ([PSZ](#))
Pointer to the fully-qualified file name.

FDM_VALIDATE Field - usSeltype

usSeltype ([USHORT](#))
Selection type.

FDM_VALIDATE Parameter - rc

rc ([BOOL](#))
Validity indicator.

TRUE	File name is valid.
FALSE	File name is not valid.

FDM_VALIDATE - Parameters

pFileName ([PSZ](#))
Pointer to the fully-qualified file name.

usSeltype ([USHORT](#))
Selection type.

rc ([BOOL](#))
Validity indicator.

TRUE	File name is valid.
FALSE	File name is not valid.

FDM_VALIDATE - Syntax

This message is sent when the user selects a file and presses Enter or clicks on the OK button, or double-clicks on a file name in the file list box.

param1

```
    PSZ      pFileName /* Pointer to the fully-qualified file name. */

param2
    USHORT   usSeltype /* Selection type. */
```

FDM_VALIDATE - Remarks

This message is only sent just before the dialog returns to the caller with the user-selected file name. Before this message is sent, *pFileName* is updated with the user-selected file name. The application can determine if this file name is acceptable. For instance, if the file dialog is being used to pick a "SaveAs" file name, the application can check to see if the file is read-only. If it is, a warning dialog should be brought up to notify the user.

When FALSE is returned from a FDM_VALIDATE message, the dialog will not be dismissed and the user can continue to use the File Dialog to select an alternate file.

In multiple file selection dialogs this message is sent for each selected entry within the file list box. When the name of the file being validated comes from a selected entry in the list box, *param2* will contain FDS_LBSELECTION. When the name of the file comes from the file name entry field, *param2* will contain FDS_EFSELECTION. Single file selection dialogs will always return FDS_EFSELECTION in *param2* since the returned file name always comes from the single line entry field.

FDM_VALIDATE - Default Processing

The [WinDefDlgProc](#) function does not expect to receive this message and takes no action on it other than to return FALSE.

FDM_VALIDATE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

Default Font Dialog Processing

This section describes how messages are processed by the default dialog procedure of the font dialog. This standard dialog can be used to provide a common, consistent font selection function.

The font dialog's default procedure can be called using the WinDefFontDlgProc function. A user-provided subclassing dialog procedure should make this call for all messages that it does not process when using the font dialog.

The default dialog procedure of the font dialog sends the messages listed in this section to itself to perform the requested action. This design allows a user-provided dialog procedure to customize the font dialog to its own needs.

Font Dialog Messages

This section describes the font dialog procedure actions on receiving the following messages.

WM_DRAWITEM (in Font Dialog)

WM_DRAWITEM (in Font Dialog) Field - id

id ([USHORT](#))
Window identifier.

The window ID of the sample area (DID_SAMPLE).

WM_DRAWITEM (in Font Dialog) Field - pOwnerItem

pOwnerItem ([POWNERITEM](#))
Pointer to an [OWNERITEM](#) data structure.

The following list defines the [OWNERITEM](#) data structure fields as they apply to the font dialog. See [OWNERITEM](#) for the default field values.

<i>hwnd</i> (HWND)	Window handle of the sample area.
<i>hps</i> (HPS)	Presentation-space handle.
<i>fsState</i> (ULONG)	Reserved.
<i>fsAttribute</i> (ULONG)	Reserved.
<i>fsStateOld</i> (ULONG)	Reserved.
<i>fsAttributeOld</i> (ULONG)	Reserved.
<i>rcItem</i> (RECTL)	Item rectangle to be drawn in window coordinates.
<i>idItem</i> (LONG)	Reserved.
<i>hItem</i> (CNRDRAWITEMINFO)	Reserved.

WM_DRAWITEM (in Font Dialog) Return Value - rc

rc ([BOOL](#))

Item-drawn indicator.

TRUE

The owner draws the item.

FALSE

If the owner does not draw the item, the owner returns this value and the font dialog draws the item.

WM_DRAWITEM (in Font Dialog) - Parameters

id ([USHORT](#))

Window identifier.

The window ID of the sample area (DID_SAMPLE).

pOwnerItem ([POWNERITEM](#))

Pointer to an [OWNERITEM](#) data structure.

The following list defines the [OWNERITEM](#) data structure fields as they apply to the font dialog. See [OWNERITEM](#) for the default field values.

hwnd ([HWND](#)) Window handle of the sample area.

hps ([HPS](#)) Presentation-space handle.

fsState ([ULONG](#)) Reserved.

fsAttribute ([ULONG](#))
Reserved.

fsStateOld ([ULONG](#))
Reserved.

fsAttributeOld ([ULONG](#))
Reserved.

rcItem ([RECTL](#)) Item rectangle to be drawn in window coordinates.

idItem ([LONG](#)) Reserved.

hItem ([CNRDRAWITEMINFO](#))
Reserved.

rc ([BOOL](#))

Item-drawn indicator.

TRUE

The owner draws the item.

FALSE

If the owner does not draw the item, the owner returns this value and the font dialog draws the item.

WM_DRAWITEM (in Font Dialog) - Syntax

If the `FNTS_OWNERDRAWPREVIEW` style is set for a font dialog, this notification message is sent to that dialog's owner whenever the preview window area (sample text) is to be drawn.

```
param1
    USHORT        id            /* Window identifier. */

param2
    POWNERITEM    pOwnerItem    /* Pointer to an OWNERITEM data structure. */
```

WM_DRAWITEM (in Font Dialog) - Remarks

The font dialog provides this message to give the application the opportunity to provide a custom drawn preview area.

The font dialog default dialog procedure generates this message and sends it to its owner, informing the owner that the preview area is to be drawn. The owner is then given the opportunity to draw that area and to indicate that the area has been drawn or that the font dialog is to draw it.

WM_DRAWITEM (in Font Dialog) - Default Processing

For a description of the default processing, see [WM_DRAWITEM](#).

WM_DRAWITEM (in Font Dialog) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

FNTM_FACENAMECHANGED

FNTM_FACENAMECHANGED Field - pFamilyname

pFamilyname ([PSZ](#))
Pointer to the currently-selected face name.

FNTM_FACENAMECHANGED Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

FNTM_FACENAMECHANGED Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

FNTM_FACENAMECHANGED - Parameters

pFamilyname (PSZ)
Pointer to the currently-selected face name.

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

FNTM_FACENAMECHANGED - Syntax

This message notifies the subclassing application whenever the font family name is changed by the user.

```
param1
    PSZ    pFamilyname /* Pointer to the currently-selected face name. */

param2
    ULONG  ulReserved  /* Reserved value, should be 0. */
```

FNTM_FACENAMECHANGED - Remarks

pFamilyname is the currently selected family name. The application can modify this string if it desires. The buffer set aside is the maximum size a face name string can be (FACESIZE).

FNTM_FACENAMECHANGED - Default Processing

The [WinDefDlgProc](#) function does not expect to receive this message and takes no action on it other than to return 0.

FNTM_FACENAMECHANGED - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

FNTM_FILTERLIST

FNTM_FILTERLIST Field - pFontname

pFontname ([PSZ](#))
Pointer to the text string that is being added to the combination box.

FNTM_FILTERLIST Field - usFieldId

usFieldId ([USHORT](#))
Field identifier.

The identifier of the field to which the text string is being added. The identifier can be one of the following:

- FNTI_FAMILYNAME**
The text string is an addition to the family name combination box.
- FNTI_STYLENAME**
The text string is an addition to the style combination box.
- FNTI_POINTSIZE**
The text string is an addition to the size combination box.

FNTM_FILTERLIST Field - usFontType

usFontType (USHORT)
Font information.

The family name, style, or point size that is being added to the combination box. Use one of the following to identify the font information that is being added:

FNTI_BITMAPFONT
A bit-map font is being added or a point size of a bit-map font is being added.

FNTI_VECTORFONT
A vector font is being added.

FNTI_SYNTHESIZED
A synthesized font is being added. This value is valid for the style field only.

FNTI_FIXEDWIDTHFONT
A fixed width (monospace) font is being added.

FNTI_PROPORTIONALFONT
A proportionally spaced font is being added.

FNTI_DEFAULTLIST
A point size from the default list (or the application-supplied list) is being added.

FNTM_FILTERLIST Return Value - rc

rc (BOOL)
Filter indicator.

TRUE
Add the text string to the combination box.

FALSE
Do not add the text string to the combination box.

FNTM_FILTERLIST - Parameters

pFontname (PSZ)
Pointer to the text string that is being added to the combination box.

usFieldId (USHORT)
Field identifier.

The identifier of the field to which the text string is being added. The identifier can be one of the following:

FNTI_FAMILYNAME
The text string is an addition to the family name combination box.

FNTI_STYLENAME
The text string is an addition to the style combination box.

FNTI_POINTSIZE
The text string is an addition to the size combination box.

usFontType (USHORT)
Font information.

The family name, style, or point size that is being added to the combination box. Use one of the following to identify the font information that is being added:

FNTI_BITMAPFONT	A bit-map font is being added or a point size of a bit-map font is being added.
FNTI_VECTORFONT	A vector font is being added.
FNTI_SYNTHESIZED	A synthesized font is being added. This value is valid for the style field only.
FNTI_FIXEDWIDTHFONT	A fixed width (monospace) font is being added.
FNTI_PROPORTIONALFONT	A proportionally spaced font is being added.
FNTI_DEFAULTLIST	A point size from the default list (or the application-supplied list) is being added.
rc (BOOL)	Filter indicator.
TRUE	Add the text string to the combination box.
FALSE	Do not add the text string to the combination box.

FNTM_FILTERLIST - Syntax

This message is sent whenever the Font Dialog is preparing to add a font family name, font style type, or point size entry to the combination box fields that contain these parameters.

```
param1
    PSZ      pFontname    /* Pointer to the text string that is being added to the combination box. */

param2
    USHORT   usFieldId     /* Field identifier. */
    USHORT   usFontType    /* Font information. */
```

FNTM_FILTERLIST - Remarks

The application checks this message to obtain the name and the .TYPE EA extended attribute of the file being added. The application then determines whether or not the file will be added.

When FALSE is returned, the file is not added to the dialog's list box.

FNTM_FILTERLIST - Default Processing

The [WinDefDlgProc](#) function does not expect to receive this message and takes no action on it other than to return FALSE.

FNTM_FILTERLIST - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

FNTM_POINTSIZECHANGED

FNTM_POINTSIZECHANGED Field - pPointSize

pPointSize ([PSZ](#))

Pointer to the text in the point-size entry field.

FNTM_POINTSIZECHANGED Field - fxPointSize

fxPointSize ([FIXED](#))

Point size.

The *fxPointSize* field in [FONTDLG](#) stated in fixed-point notation.

FNTM_POINTSIZECHANGED Return Value - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

FNTM_POINTSIZECHANGED - Parameters

pPointSize ([PSZ](#))

Pointer to the text in the point-size entry field.

fxPointSize ([FIXED](#))
Point size.

The *fxPointSize* field in [FONTDLG](#) stated in fixed-point notation.

ulReserved ([ULONG](#))
Reserved value, should be 0.

FNTM_POINTSIZECHANGED - Syntax

This message notifies subclassing applications when the point size of the font is changed by the user.

```
param1
    PSZ      pPointSize /* Pointer to the text in the point-size entry field. */

param2
    FIXED fxPointSize /* Point size. */
```

FNTM_POINTSIZECHANGED - Remarks

When the application wants to limit the point sizes the user can select, it should process this message by changing the *pPointSize* value and putting up a message box explaining the limitation to the user.

FNTM_POINTSIZECHANGED - Default Processing

The [WinDefDlgProc](#) function does not expect to receive this message and takes no action on it other than to return 0.

FNTM_POINTSIZECHANGED - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

FNTM_STYLECHANGED

FNTM_STYLECHANGED Field - styc

styc ([STYLECHANGE](#))
Style changes.

FNTM_STYLECHANGED Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

FNTM_STYLECHANGED Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

FNTM_STYLECHANGED - Parameters

styc ([STYLECHANGE](#))
Style changes.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

FNTM_STYLECHANGED - Syntax

This message notifies subclassing applications when the user changes any of the attributes in the [STYLECHANGE](#) structure.

```
param1  
    STYLECHANGE styc    /* Style changes. */
```

```
param2
    ULONG        ulReserved /* Reserved value, should be 0. */
```

FNTM_STYLECHANGED - Remarks

The "Old" fields show the style attributes before the user made the change. The other parameters show what the state will be after the application passes this message to [WinDefFontDlgProc](#). When the "Old" field and the "New" field are the same, no change is made for that attribute.

FNTM_STYLECHANGED - Default Processing

The [WinDefDlgProc](#) function does not expect to receive this message and takes no action on it other than to return 0.

FNTM_STYLECHANGED - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

FNTM_UPDATEPREVIEW

FNTM_UPDATEPREVIEW Field - hwndPreview

hwndPreview ([HWND](#))
Window handle.

Window handle the preview image is drawn into. This is a static text field.

FNTM_UPDATEPREVIEW Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

FNTM_UPDATEPREVIEW Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

FNTM_UPDATEPREVIEW - Parameters

hwndPreview ([HWND](#))
Window handle.

Window handle the preview image is drawn into. This is a static text field.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

FNTM_UPDATEPREVIEW - Syntax

This message notifies subclassing applications before the preview window is updated. This occurs when the font selection is modified.

```
param1
    HWND    hwndPreview    /* Window handle. */

param2
    ULONG    ulReserved    /* Reserved value, should be 0. */
```

FNTM_UPDATEPREVIEW - Remarks

This message notifies an application that the dialog is about to update the preview area.

FNTM_UPDATEPREVIEW - Default Processing

The [WinDefDlgProc](#) function does not expect to receive this message and takes no action on it other than to return 0.

FNTM_UPDATEPREVIEW - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

Direct Manipulation (Drag) Message Processing

This section describes the processing that occurs during a direct manipulation operation when the application sends or receives a direct manipulation (DM_*) message.

Direct Manipulation Messages

This section describes messages that an application may send or receive during a direct manipulation operation.

DM_DISCARDOBJECT

DM_DISCARDOBJECT Field - pDragInfo

pDragInfo ([PDRAGINFO](#))

Pointer to the [DRAGINFO](#) structure representing the items to be discarded.

DM_DISCARDOBJECT Field - ulReserved

ulReserved ([MPARAM](#))

Reserved value, should be NULL.

DM_DISCARDOBJECT Return Value - ulAction

ulAction (ULONG)	Flag.
DRR_SOURCE	The source window procedure accepts responsibility for the operation.
DRR_TARGET	The target window procedure is to accept responsibility for the operation. The OS/2 shell supports the discarding of dragitems that can be rendered by the DRM_OS2FILE method.
DRR_ABORT	Abort the entire DM_DROP action.

DM_DISCARDOBJECT - Parameters

pDragInfo (PDRAGINFO)	Pointer to the DRAGINFO structure representing the items to be discarded.
ulReserved (MPARAM)	Reserved value, should be NULL.
ulAction (ULONG)	Flag.
DRR_SOURCE	The source window procedure accepts responsibility for the operation.
DRR_TARGET	The target window procedure is to accept responsibility for the operation. The OS/2 shell supports the discarding of dragitems that can be rendered by the DRM_OS2FILE method.
DRR_ABORT	Abort the entire DM_DROP action.

DM_DISCARDOBJECT - Syntax

This message is sent to a source that supports the "DRM_DISCARD" rendering method.

```
param1
    PDRAGINFO pDragInfo /* Pointer to the DRAGINFO structure representing the items to be discarded. */
mpparam2
    MPARAM      ulReserved /* Reserved value, should be NULL. */
```

DM_DISCARDOBJECT - Remarks

This message is sent to the source window for the drag action. The source should make a copy of the parameters and return. The source

should also create a separate thread to execute the discard action if it responds with DRR_SOURCE.

DM_DISCARDOBJECT - Default Processing

The [WinDefWindowProc](#) function does not expect to receive this message and takes no action on it, other than to set *ulAction* to the default value of NULL.

DM_DISCARDOBJECT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

DM_DRAGERROR

DM_DRAGERROR Field - usError

usError ([USHORT](#))
Error code.

Returned from DosCopy, DosMove, or DosDelete.

DM_DRAGERROR Field - usOperation

usOperation ([USHORT](#))
Flag.

Flag indicating the operation that failed.

DFF_MOVE	DosMove failed.
DFF_COPY	DosCopy failed.
DFF_DELETE	DosDelete failed.

DM_DRAGERROR Field - hstr

hstr ([HSTR](#))
[HSTR](#) of file contributing to the error.

DM_DRAGERROR Return Value - hstrAction

hstrAction ([HSTR](#))
Action indicator.

DME_IGNORECONTINUE
Do not retry the operation, but continue with the rest of the files.

DME_IGNOREABORT
Do not retry the operation, and do not try any other files.

DME_RETRY
Retry the operation.

DME_REPLACE
Replace the file at the destination. Used if is not specified.

Other
[HSTR](#) of new file name to use for retry.

DM_DRAGERROR - Parameters

usError ([USHORT](#))
Error code.

Returned from DosCopy, DosMove, or DosDelete.

usOperation ([USHORT](#))
Flag.

Flag indicating the operation that failed.

DFF_MOVE
DosMove failed.

DFF_COPY
DosCopy failed.

DFF_DELETE
DosDelete failed.

hstr ([HSTR](#))
[HSTR](#) of file contributing to the error.

hstrAction ([HSTR](#))
Action indicator.

DME_IGNORECONTINUE
Do not retry the operation, but continue with the rest of the files.

DME_IGNOREABORT
Do not retry the operation, and do not try any other files.

DME_RETRY
Retry the operation.

DME_REPLACE
Replace the file at the destination. Used if is not specified.

Other
[HSTR](#) of new file name to use for retry.

DM_DRAGERROR - Syntax

This message is sent to the caller of [DrgDragFiles](#) or [DrgAcceptDroppedFiles](#) when an error occurs during a move or copy operation for a file.

```
param1
    USHORT  usError      /* Error code. */
    USHORT  usOperation  /* Flag. */

param2
    HSTR     hstr        /* HSTR of file contributing to the error. */
```

DM_DRAGERROR - Remarks

The receiver of this message should return the action that the sender should take.

DM_DRAGERROR - Default Processing

The [WinDefWindowProc](#) function does not expect to receive this message and takes no action other than to return FALSE.

DM_DRAGERROR - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

DM_DRAGFILECOMPLETE

DM_DRAGFILECOMPLETE Field - hstr

hstr ([HSTR](#))
File handle.

DM_DRAGFILECOMPLETE Field - usOperation

usOperation ([USHORT](#))
Flags.

DF_MOVE The operation was a move. If this flag is not set, the operation was a copy.

DF_SOURCE The receiving window was the source of the drag. If this flag is not set, the receiver was the target of the drop.

DF_SUCCESSFUL The drag operation was successful for the file. If this flag is not set, the operation failed.

DM_DRAGFILECOMPLETE Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

DM_DRAGFILECOMPLETE - Parameters

hstr ([HSTR](#))
File handle.

usOperation ([USHORT](#))
Flags.

DF_MOVE The operation was a move. If this flag is not set, the operation was a copy.

DF_SOURCE The receiving window was the source of the drag. If this flag is not set, the receiver was the target of the drop.

DF_SUCCESSFUL

The drag operation was successful for the file. If this flag is not set, the operation failed.

ulReserved (ULONG)

Reserved value, should be 0.

DM_DRAGFILECOMPLETE - Syntax

This message is sent when a direct manipulation operation on a file or files is complete.

```
param1
    HSTR    hstr        /* File handle. */

param2
    USHORT  usOperation /* Flags. */
```

DM_DRAGFILECOMPLETE - Remarks

hstr is [HSTR](#) for the source file if this message is sent by [DrgDragFiles](#), and is [HSTR](#) for the target file if this message is sent by [DrgAcceptDroppedFiles](#).

This message is sent by [DrgDragFiles](#) to its caller when the move or copy operation is completed, regardless of success or failure. It is also sent by [DrgAcceptDroppedFiles](#) when a file has been successfully dropped on the caller.

DM_DRAGFILECOMPLETE - Default Processing

The [WinDefWindowProc](#) function does not expect to receive this message and takes no action other than to return 0.

DM_DRAGFILECOMPLETE - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

DM_DRAGLEAVE

DM_DRAGLEAVE Field - pDraginfo

pDraginfo ([PDRAGINFO](#))

Pointer to the [DRAGINFO](#) structure for the drag operation.

DM_DRAGLEAVE Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

DM_DRAGLEAVE Return Value - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

DM_DRAGLEAVE - Parameters

pDraginfo ([PDRAGINFO](#))

Pointer to the [DRAGINFO](#) structure for the drag operation.

ulReserved ([ULONG](#))

Reserved value, should be 0.

ulReserved ([ULONG](#))

Reserved value, should be 0.

DM_DRAGLEAVE - Syntax

This message is sent to a window that is being dragged over when one of these conditions occur:

- The object is dragged outside the boundaries of the window.
- The drag operation is terminated while the object is over the window.


```
param1
    PDRAGINFO pDraginfo /* Pointer to the DRAGINFO structure for the drag operation. */

param2
    ULONG      ulReserved /* Reserved value, should be 0. */
```

DM_DRAGLEAVE - Remarks

This message allows for target emphasis and de-emphasis during the direct manipulation process. This message is not sent when a drop occurs. Use [DM_DROP](#) as a signal to remove the target emphasis.

DM_DRAGLEAVE - Default Processing

The [WinDefWindowProc](#) function does not expect to receive this message and takes no action on it other than to return 0.

DM_DRAGLEAVE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

DM_DRAGOVER

DM_DRAGOVER Field - pDraginfo

pDraginfo ([PDRAGINFO](#))
Pointer to the [DRAGINFO](#) structure representing the object being dragged.

DM_DRAGOVER Field - sxDrop

syDrop ([SHORT](#))
X-coordinate of the pointing device pointer in desktop coordinates.

DM_DRAGOVER Field - syDrop

syDrop ([SHORT](#))
Y-coordinate of the pointing device pointer in desktop coordinates.

DM_DRAGOVER Field - usDrop

usDrop ([USHORT](#))
Drop indicator.

DOR_DROP
Object can be dropped. When this reply is given, *usDefaultOp* must be set to indicate which operation is performed if the user should drop at this location.

DOR_NODROP
Object cannot be dropped at this time. The target can accept the object in the specified type and format using the specified operation, but the current state of the target will not allow it to be dropped on. The target may change state in the future so that the same object may be acceptable.

DOR_NODROPOP
Object cannot be dropped at this time. The target can accept the object in the specified type and format, but the current operation is not acceptable. A change in the drag operation may change the acceptability of the object.

DOR_NEVERDROP
Object cannot be dropped. The target cannot accept the object now and will not change state so that the object will be acceptable in the future. If this response is returned, no more DM_DRAGOVER messages will be sent to the target until the pointer is moved out of and back into the target window.

DM_DRAGOVER Field - usDefaultOp

usDefaultOp ([USHORT](#))
Target-defined default operation.

DO_COPY
Default operation is a copy.

DO_LINK
Default operation is a link.

DO_MOVE
Default operation is a move.

DO_CREATE
Default operation is create. Used to create an object from a template.

DO_NEW
Default operation is create another. Use create another to create an object that has default settings and data. The result of using create another is identical to creating an object from a template. This value should be defined as

DO_UNKNOWN+3 if it is not recognized in the current level of the toolkit.

Other

Operation is defined by the application.

This value should be greater than or equal to (\geq) DO_UNKNOWN but not DO_NEW.

DM_DRAGOVER - Parameters

pDraginfo (PDRAGINFO)

Pointer to the DRAGINFO structure representing the object being dragged.

sxDrop (SHORT)

X-coordinate of the pointing device pointer in desktop coordinates.

syDrop (SHORT)

Y-coordinate of the pointing device pointer in desktop coordinates.

usDrop (USHORT)

Drop indicator.

DOR_DROP

Object can be dropped. When this reply is given, *usDefaultOp* must be set to indicate which operation is performed if the user should drop at this location.

DOR_NODROP

Object cannot be dropped at this time. The target can accept the object in the specified type and format using the specified operation, but the current state of the target will not allow it to be dropped on. The target may change state in the future so that the same object may be acceptable.

DOR_NODROPOP

Object cannot be dropped at this time. The target can accept the object in the specified type and format, but the current operation is not acceptable. A change in the drag operation may change the acceptability of the object.

DOR_NEVERDROP

Object cannot be dropped. The target cannot accept the object now and will not change state so that the object will be acceptable in the future. If this response is returned, no more DM_DRAGOVER messages will be sent to the target until the pointer is moved out of and back into the target window.

usDefaultOp (USHORT)

Target-defined default operation.

DO_COPY

Default operation is a copy.

DO_LINK

Default operation is a link.

DO_MOVE

Default operation is a move.

DO_CREATE

Default operation is create. Used to create an object from a template.

DO_NEW

Default operation is create another. Use create another to create an object that has default settings and data. The result of using create another is identical to creating an object from a template. This value should be defined as DO_UNKNOWN+3 if it is not recognized in the current level of the toolkit.

Other

Operation is defined by the application.

This value should be greater than or equal to (\geq) DO_UNKNOWN but not DO_NEW.

DM_DRAGOVER - Syntax

This message allows the window under the mouse pointer to determine if the object or objects currently being dragged can be dropped.

param2 is the pointing device pointer location.

```
param1
    PDRAGINFO  pDraginfo    /* Pointer to the DRAGINFO structure representing the object being dragged. */

param2
    SHORT      sxDrop       /* X-coordinate of the pointing device pointer in desktop coordinates. */
    SHORT      syDrop       /* Y-coordinate of the pointing device pointer in desktop coordinates. */

returns
    USHORT     usDrop        /* Drop indicator. */
    USHORT     usDefaultOp   /* Target-defined default operation. */
```

DM_DRAGOVER - Remarks

This message is sent to the window that is directly under the hot spot of the mouse pointer during the drag operation when any of the following conditions are met:

- The user moves the mouse.
- A key is pressed.
- A [WM_BUTTON1UP](#), [WM_BUTTON2UP](#), [WM_BUTTON3UP](#), or [WM_ENDDRAG](#) message is received. The message corresponds to parameter specified by the call to [DrgDrag](#) indicating that the drag is ending. In this case the message is sent only if the mouse has moved since the last [DM_DRAGOVER](#) message was sent.

The receiver can gain access to the [DRAGINFO](#) structure with [DrgAccessDraginfo](#). The acceptability of the dragged objects can be determined by querying the *hstrType* and *hstrRMF* string handles in each of the [DRAGITEM](#) structures carried in [DRAGINFO](#) structure. In order to accept the drop, the target window must be able to accept *all* of the objects that are being dragged.

The receiver should provide target emphasis for itself. The receiver can use [DrgSetDragPointer](#) to change the bit map while it is being dragged over. A [DM_DRAGLEAVE](#) or [DM_DROP](#) message will be sent to the target in the future. Target emphasis should be removed at that time.

If *usOperation* in [DRAGINFO](#) is [DO_DEFAULT](#) or [DO_UNKNOWN](#) and the target returns [DOR_DROP](#) for *usDrop*, *usDefaultOp* should be set to reflect what the target defines as the default operation. This information is used to provide the appropriate modification to the drag pointer and the target's default operation will be passed in the *usOperation* field of the [DRAGINFO](#) structure specified in the [DM_DROP](#) message.

If the value of the *usOperation* field is not [DO_DEFAULT](#) or [DO_UNKNOWN](#), the *usDefaultOp* parameter is ignored.

Note: Lazy drag enabled applications are expected to process this message. It is to be handled in the same manner as the standard drag enabled applications.

DM_DRAGOVER - Default Processing

The [WinDefWindowProc](#) function returns [DOR_NEVERDROP](#) to the sender of this message.

DM_DRAGOVER - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

DM_DRAGOVERNOTIFY

DM_DRAGOVERNOTIFY Field - pDraginfo

pDraginfo ([PDRAGINFO](#))

Pointer to the [DRAGINFO](#) structure that represents the object being dragged.

DM_DRAGOVERNOTIFY Field - usDrop

usDrop ([USHORT](#))

Drop indicator.

DM_DRAGOVERNOTIFY Field - usDefaultOp

usDefaultOp ([USHORT](#))

Default operation.

Target-defined default operation.

DM_DRAGOVERNOTIFY Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value.

DM_DRAGOVERNOTIFY - Parameters

pDraginfo ([PDRAGINFO](#))
Pointer to the [DRAGINFO](#) structure that represents the object being dragged.

usDrop ([USHORT](#))
Drop indicator.

usDefaultOp ([USHORT](#))
Default operation.

Target-defined default operation.

ulReserved ([ULONG](#))
Reserved value.

DM_DRAGOVERNOTIFY - Syntax

This message is sent to the source of a drag operation immediately after a [DM_DRAGOVER](#) message is sent to a target window.

param2 is the target's reply to the [DM_DRAGOVER](#) message.

```
param1
    PDRAGINFO    pDraginfo    /* Pointer to the DRAGINFO structure that represents the object being dragged. */

param2
    USHORT        usDrop        /* Drop indicator. */
    USHORT        usDefaultOp   /* Default operation. */
```

DM_DRAGOVERNOTIFY - Remarks

The source window can use this message to modify its behavior or appearance based on a target window's response to the [DM_DRAGOVER](#) message.

See [DM_DRAGOVER](#) for a description of the target window's possible responses.

DM_DRAGOVERNOTIFY - Default Processing

The [WinDefWindowProc](#) function does not expect to receive this message and therefore takes no action on it other than to return NULL.

DM_DRAGOVERNOTIFY - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

DM_DROP

DM_DROP Field - pDraginfo

pDraginfo ([PDRAGINFO](#))
Pointer to the [DRAGINFO](#) structure.

DM_DROP Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

DM_DROP Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

DM_DROP - Parameters

pDraginfo ([PDRAGINFO](#))
Pointer to the [DRAGINFO](#) structure.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

DM_DROP - Syntax

This message is sent to the target when the dragged object is dropped.

```
param1
    PDRAGINFO   pDraginfo   /* Pointer to the DRAGINFO structure. */

param2
    ULONG        ulReserved /* Reserved value, should be 0. */
```

DM_DROP - Remarks

This message is sent to the target window directly under the hot spot of the mouse pointer at the completion of a direct-manipulation operation only if DOR_DROP was returned for the [DM_DRAGOVER](#) message sent to the target window during the drag.

The receiver can obtain access to [DRAGINFO](#) structure with [DrgAccessDraginfo](#).

The receiver must immediately remove any target emphasis and post a private message to itself to initiate the data transfer conversations needed to complete the operation.

The receiver can use the *cxOffset* and *cyOffset* fields in the [DRAGITEM](#) structure to position the dropped object within its window relative to the drop point. Multiple objects are moved in the same relative position to each other in the target window as they were in the source.

With standard drag, [DrgDrag](#) does not return until the drag set is dropped on a target window. Since the source window is the caller of the [DrgDrag](#), it receives the handle of the target window that the drag set is dropped on when [DrgDrag](#) returns.

Lazy Drag is slightly different. Since the drag operation is non-modal, the [DrgLazyDrag](#) returns as soon as it has completed its initialization of the drag. [DM_DROPNOTIFY](#) is posted to the source window after the drag set is dropped.

When the application receiving the DM_DROP message has finished all data transfer operations, the target window must free the [DRAGINFO](#) structure using [DrgFreeDraginfo](#).

DM_DROP - Default Processing

The [WinDefWindowProc](#) function calls [DrgDeleteDraginfoStrHandles](#) and [DrgFreeDraginfo](#) for *pDraginfo* and returns 0.

DM_DROP - Topics

Select an item:
[Syntax](#)

DM_DROPHELP

DM_DROPHELP Field - pDraginfo

pDraginfo ([PDRAGINFO](#))

Pointer to the [DRAGINFO](#) structure used in the drag operation.

DM_DROPHELP Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

DM_DROPHELP Return Value - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

DM_DROPHELP - Parameters

pDraginfo ([PDRAGINFO](#))

Pointer to the [DRAGINFO](#) structure used in the drag operation.

ulReserved ([ULONG](#))

Reserved value, should be 0.

ulReserved ([ULONG](#))

Reserved value, should be 0.

DM_DROPHELP - Syntax

This message requests help for the current drag operation.

```
param1
    PDRAGINFO pDraginfo /* Pointer to the DRAGINFO structure used in the drag operation. */

param2
    ULONG      ulReserved /* Reserved value, should be 0. */
```

DM_DROPHELP - Remarks

This message is posted to the target of a drop when F1 is pressed during a direct-manipulation operation, and the drag operation is canceled.

The *usOperation* field of *pDraginfo* can be used to provide help information in the context of the drag operation during which it was requested.

The DM_DROPHELP message is not supported for lazy drag operations. Since the drag operation is non-modal, the user may request help on anything at any time during the drag. If the application wishes to provide drop help, it must specify the action required to invoke drop help (for example a menu choice), and code the support for it explicitly.

DM_DROPHELP - Default Processing

The `WinDefWindowProc` function calls `DrgDeleteDraginfoStrHandles` and `DrgFreeDraginfo` for *pDraginfo* and returns 0.

DM_DROPHELP - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

DM_DROPNOTIFY

DM_DROPNOTIFY Field - pDraginfo

pDraginfo ([PDRAGINFO](#))

Pointer to the [DRAGINFO](#) structure allocated by the source window receiving the message.

DM_DROPNOTIFY Field - hwndTarget

hwndTarget ([HWND](#))

Handle of the target window that the drag set was dropped on.

Note: If *hwndTarget* is equal to zero, the drag is canceled, and the drag set is not dropped. [DrgCancelLazyDrag](#) posts a DM_DROPNOTIFY message with an *hwndTarget* value of zero to the source window.

DM_DROPNOTIFY Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, must be 0.

DM_DROPNOTIFY - Parameters

pDraginfo ([PDRAGINFO](#))

Pointer to the [DRAGINFO](#) structure allocated by the source window receiving the message.

hwndTarget ([HWND](#))

Handle of the target window that the drag set was dropped on.

Note: If *hwndTarget* is equal to zero, the drag is canceled, and the drag set is not dropped. [DrgCancelLazyDrag](#) posts a DM_DROPNOTIFY message with an *hwndTarget* value of zero to the source window.

ulReserved ([ULONG](#))

Reserved value, must be 0.

DM_DROPNOTIFY - Syntax

This message provides the source window with the target window handle and a pointer to the [DRAGINFO](#) structure allocated by the source window.

```
param1
    PDRAGINFO  pDraginfo    /* Pointer to the DRAGINFO structure allocated by the source window receiving the

param2
    HWND       hwndTarget    /* Handle of the target window that the drag set was dropped on. */

returns
    ULONG      ulReserved    /* Reserved value, must be 0. */
```

DM_DROPNOTIFY - Remarks

This message is posted to the source window involved in the drag operation when the drag set is dropped on a valid target window.

The source window must examine *hwndTarget* to determine if the target window is the same as the source window. If it is not, the source window must immediately free the [DRAGINFO](#); if the source and target windows are the same, the [DRAGINFO](#) must be freed by the target window after completing the post-drop conversation.

Note: Lazy drag enabled applications are expected to process this message; standard drag applications are not.

DM_DROPNOTIFY - Default Processing

The default message procedure sets *ulReserved* to 0.

DM_DROPNOTIFY - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

DM_EMPHASIZETARGET

DM_EMPHASIZETARGET Field - sx

sx ([SHORT](#))
X-coordinate.

X-coordinate of the pointing device pointer in window coordinates.

DM_EMPHASIZETARGET Field - sy

sy ([SHORT](#))
Y-coordinate.

Y-coordinate of the pointing device pointer in window coordinates.

DM_EMPHASIZETARGET Field - usEmphasis

usEmphasis ([USHORT](#))
Flags.

TRUE Apply emphasis.
FALSE Remove emphasis.

DM_EMPHASIZETARGET Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

DM_EMPHASIZETARGET - Parameters

sx ([SHORT](#))
X-coordinate.

X-coordinate of the pointing device pointer in window coordinates.

sy ([SHORT](#))
Y-coordinate.

Y-coordinate of the pointing device pointer in window coordinates.

usEmphasis ([USHORT](#))
Flags.

TRUE Apply emphasis.

FALSE

Remove emphasis.

ulReserved ([ULONG](#))

Reserved value, should be 0.

DM_EMPHASIZETARGET - Syntax

This message is sent to the caller of [DrgAcceptDroppedFiles](#) to inform it to either apply or remove target emphasis from itself.

```
param1
    SHORT    sx           /* X-coordinate. */
    SHORT    sy           /* Y-coordinate. */

usparam2
    USHORT   usEmphasis /* Flags. */
```

DM_EMPHASIZETARGET - Default Processing

The [WinDefWindowProc](#) function does not expect to receive this message and takes no action other than to return 0.

DM_EMPHASIZETARGET - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Default Processing](#)

[Glossary](#)

DM_ENDCONVERSATION

DM_ENDCONVERSATION Field - ullItemID

ullItemID ([ULONG](#))

Item ID.

The *ulItemID* from the [DRAGITEM](#) that was contained within the [DRAGINFO](#) structure when the object was dropped.

DM_ENDCONVERSATION Field - ulFlags

ulFlags ([ULONG](#))
Flags.

The flags are set as follows:

- DMFL_TARGETSUCCESSFUL
The target successfully completed its portion of the rendering operation.
- DMFL_TARGETFAIL
The target failed to complete its portion of the rendering operation.

DM_ENDCONVERSATION Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

DM_ENDCONVERSATION - Parameters

ulItemID ([ULONG](#))
Item ID.

The *ulItemID* from the [DRAGITEM](#) that was contained within the [DRAGINFO](#) structure when the object was dropped.

ulFlags ([ULONG](#))
Flags.

The flags are set as follows:

- DMFL_TARGETSUCCESSFUL
The target successfully completed its portion of the rendering operation.
- DMFL_TARGETFAIL
The target failed to complete its portion of the rendering operation.

ulReserved ([ULONG](#))
Reserved value, should be 0.

DM_ENDCONVERSATION - Syntax

The target uses this message to notify a source that a drag operation is complete.

```
param1
    ULONG    ulItemID    /* Item ID. */

param2
    ULONG    ulFlags     /* Flags. */
```

DM_ENDCONVERSATION - Remarks

This message is used to inform a source that the target has completed its part of a rendering operation. It is sent by the target to the source.

The target must send this message under any of the following circumstances:

- The target receives a [DM_RENDERCOMPLETE](#) message and will not retry the operation.
- The target completes the rendering operation without involvement from the source.
- The target wants to terminate a rendering operation in progress.
- The target chooses not to render an object that was dropped on it.

DM_ENDCONVERSATION - Default Processing

The [WinDefWindowProc](#) function does not expect to receive this message and takes no action other than to return 0.

DM_ENDCONVERSATION - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

DM_FILERENDERED

DM_FILERENDERED Field - rndf

rndf ([PRENDERFILE](#))
Pointer to a [RENDERFILE](#) structure.

DM_FILERENDERED Field - usOperation

usOperation ([USHORT](#))
Flags.

TRUE	Operation succeeded
FALSE	Operation failed.

DM_FILERENDERED Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

DM_FILERENDERED - Parameters

rndf ([PRENDERFILE](#))
Pointer to a [RENDERFILE](#) structure.

usOperation ([USHORT](#))
Flags.

TRUE	Operation succeeded
FALSE	Operation failed.

ulReserved ([ULONG](#))
Reserved value, should be 0.

DM_FILERENDERED - Syntax

This message is sent to the window handling the drag conversation for the caller of [DrgDragFiles](#).

```
param1
    PRENDERFILE    rndf           /* Pointer to a RENDERFILE structure. */

param2
    USHORT          usOperation    /* Flags. */
```

DM_FILERENDERED - Remarks

This message is sent when the rendering (moving or copying) of a file is complete. The handle of this window is the *hwndDragFiles* field of the [RENDERFILE](#) structure sent on [DM_RENDERFILE](#).

DM_FILERENDERED - Default Processing

The [WinDefWindowProc](#) function does not expect to receive this message and takes no action other than to return 0.

DM_FILERENDERED - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

DM_PRINTOBJECT

DM_PRINTOBJECT Field - pDragItem

pDragItem ([PDRAGITEM](#))

Pointer to the [DRAGITEM](#) structure representing the objects to be printed.

DM_PRINTOBJECT Field - pPrintDest

pPrintDest ([PPRINTDEST](#))

Pointer to the [PRINTDEST](#) structure representing printer object to print to.

The structure contains all the parameters required to call the functions [DevPostDeviceModes](#) and [DevOpenDC](#).

DM_PRINTOBJECT Return Value - ulAction

ulAction ([ULONG](#))
Flag.

DRR_SOURCE	The source window procedure/object procedure will take responsibility for the print operation.
DRR_TARGET	The target printer object will take responsibility for the print operation (this will only work on objects which are of the pre-registered rendering method; "DRM_OS2FILE").
DRR_ABORT	Abort the entire DM_DROP action (do not send any more DM_PRINTOBJECT messages to any selected source object involved in this DM_DROP).

DM_PRINTOBJECT - Parameters

pDragItem ([PDRAGITEM](#))
Pointer to the [DRAGITEM](#) structure representing the objects to be printed.

pPrintDest ([PPRINTDEST](#))
Pointer to the [PRINTDEST](#) structure representing printer object to print to.

The structure contains all the parameters required to call the functions [DevPostDeviceModes](#) and [DevOpenDC](#).

ulAction ([ULONG](#))
Flag.

DRR_SOURCE	The source window procedure/object procedure will take responsibility for the print operation.
DRR_TARGET	The target printer object will take responsibility for the print operation (this will only work on objects which are of the pre-registered rendering method; "DRM_OS2FILE").
DRR_ABORT	Abort the entire DM_DROP action (do not send any more DM_PRINTOBJECT messages to any selected source object involved in this DM_DROP).

DM_PRINTOBJECT - Syntax

This message is posted to a source that supports the "DRM_PRINT" rendering method when objects are dropped on a printer object.

```
param1
    PDRAGITEM    pDragItem    /* Pointer to the DRAGITEM structure representing the objects to be printed. */
param2
    PPRINTDEST    pPrintDest    /* Pointer to the PRINTDEST structure representing printer object to print to. */
```

DM_PRINTOBJECT - Remarks

This message is posted to the source window procedure. The source window procedure is responsible for interpreting the structure given by *param2*. It should make a copy of all the parameters and then return.

The receiver of this message should create a thread in which to dispatch this message in order to facilitate a prompt reply. The thread can then call [DevPostDeviceModes](#) and [DevOpenDC](#) as appropriate.

Note: Technically, the message is sent, but it is from a spun-off thread so the effect is that of a posted message - it is not synchronous.

All the drag and drop protocol messages occur synchronously during the [DrgDrag](#) call. If the DM_PRINTOBJECT message is returned to the application asynchronously, by the time the application gets the message, the [DRAGINFO](#) structure (and [DRAGITEM](#) inside) have already been freed. The application will trap when it tries to use *param1* of the DM_PRINTOBJECT message (the pointer to the [DRAGITEM](#) data structure).

For debugging purposes, the application can keep a static copy of the [DRAGINFO](#) pointer and use [DrgAccessDraginfo](#) to reaccess the shared [DRAGINFO](#) structure during DM_PRINTOBJECT message processing.

DM_PRINTOBJECT - Default Processing

The [WinDefWindowProc](#) function does not expect to receive this message and takes no action on it, other than to set *ulAction* to the default value of NULL.

DM_PRINTOBJECT - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

DM_RENDER

DM_RENDER Field - pDxfer

pDxfer ([PDRAGTRANSFER](#))

Pointer to the [DRAGTRANSFER](#) structure.

DM_RENDER Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

DM_RENDER Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DM_RENDER - Parameters

pDxfer ([PDRAGTRANSFER](#))
Pointer to the [DRAGTRANSFER](#) structure.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DM_RENDER - Syntax

This message is used to request a source to provide a rendering of an object in a specified rendering mechanism and format.

```
param1
    PDRAGTRANSFER    pDxfer        /* Pointer to the DRAGTRANSFER structure. */

param2
    ULONG              ulReserved    /* Reserved value, should be 0. */
```

DM_RENDER - Remarks

The target sends this message to a source window to request a rendering of an object. If the source returns FALSE, it may set flags in the [DRAGTRANSFER](#) structure that tell the target how to perform the rendering operation on its own, or how to retry the operation. If no flags are set, the source will not allow a rendering of the object.

If TRUE is returned, the message was processed by the recipient and the requested rendering will take place. The source will post a [DM_RENDERCOMPLETE](#) message to the target when the rendering is complete.

If FALSE is returned, either the message was not processed by the recipient, or the recipient could not perform the requested rendering. See *IsReply* in [DRAGTRANSFER](#) for more information.

DM_RENDER - Default Processing

The [WinDefWindowProc](#) function does not expect to receive this message and takes no action other than to return 0.

DM_RENDER - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

DM_RENDERCOMPLETE

DM_RENDERCOMPLETE Field - pDxfer

pDxfer ([PDRAGTRANSFER](#))

Pointer to the [DRAGTRANSFER](#) structure.

This must be the same pointer that was sent to the source in the [DM_RENDER](#) message.

DM_RENDERCOMPLETE Field - usFS

- usFS (USHORT)**
Flag field.
- Flag field indicating successful completion.
- DMFL_RENDERFAIL**
The source is unable to perform the rendering operation. The target may be allowed to retry. If the target is allowed to retry and chooses not to, it must send a **DM_ENDCONVERSATION** message to the source.
- DMFL_RENDEROK**
The source has completed the rendering operation. When the target completes its part of the rendering operation, it must post a **DM_RENDERCOMPLETE** message to the source.
- DMFL_RENDERRETRY**
The source has completed the rendering operation and will allow the target to retry its part of the operation if it fails. This flag can be set in conjunction with either the **DMFL_RENDERFAIL** or **DMFL_RENDEROK** flags.
-

DM_RENDERCOMPLETE Return Value - ulReserved

- ulReserved (ULONG)**
Reserved value, should be 0.
-

DM_RENDERCOMPLETE - Parameters

- pDxfer (PDRAGTRANSFER)**
Pointer to the **DRAGTRANSFER** structure.
- This must be the same pointer that was sent to the source in the **DM_RENDER** message.
- usFS (USHORT)**
Flag field.
- Flag field indicating successful completion.
- DMFL_RENDERFAIL**
The source is unable to perform the rendering operation. The target may be allowed to retry. If the target is allowed to retry and chooses not to, it must send a **DM_ENDCONVERSATION** message to the source.
- DMFL_RENDEROK**
The source has completed the rendering operation. When the target completes its part of the rendering operation, it must post a **DM_RENDERCOMPLETE** message to the source.
- DMFL_RENDERRETRY**
The source has completed the rendering operation and will allow the target to retry its part of the operation if it fails. This flag can be set in conjunction with either the **DMFL_RENDERFAIL** or **DMFL_RENDEROK** flags.
- ulReserved (ULONG)**
Reserved value, should be 0.
-

DM_RENDERCOMPLETE - Syntax

This message is posted by a source to a target window. It informs the target that the source has completed a requested rendering operation.

```
param1
    PDRAGTRANSFER  pDxfer      /* Pointer to the DRAGTRANSFER structure. */

param2
    USHORT          usFS        /* Flag field. */
```

DM_RENDERCOMPLETE - Remarks

If the rendering operation failed for any reason, the source can allow the target to retry the operation. The source should return to the state it was in when the drop occurred for that object. The target resumes the rendering operation from the beginning.

If the rendering operation encounters a permanent failure, the source should fail the operation and proceed as if the rendering was completed.

If the rendering operation completes successfully, the source should return to the state it was in when the drop occurred for that object. This allows the target to retry the operation if its portion of the rendering failed. The target must post a [DM_ENDCONVERSATION](#) message when either of the following occurs:

- It determines that the rendering operation successfully completed
 - It chooses not to retry a rendering operation that failed.
-

DM_RENDERCOMPLETE - Default Processing

The [WinDefWindowProc](#) function should send a [DM_ENDCONVERSATION](#) message to the window indicated in the *hwndItem* field of the [DRAGITEM](#) structure. The message should indicate that the target failed in its part of the rendering operation. Sending the [DM_ENDCONVERSATION](#) message allows the source to release the resources it dedicated to the rendering operation.

DM_RENDERCOMPLETE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

DM_RENDERFILE

DM_RENDERFILE Field - rndf

rndf ([PRENDERFILE](#))
Pointer to a [RENDERFILE](#) structure.

DM_RENDERFILE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

DM_RENDERFILE Return Value - rc

rc ([BOOL](#))
Render handling.

TRUE	The receiver handled the rendering.
FALSE	DrgDragFiles should render this file.

DM_RENDERFILE - Parameters

rndf ([PRENDERFILE](#))
Pointer to a [RENDERFILE](#) structure.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Render handling.

TRUE	The receiver handled the rendering.
FALSE	DrgDragFiles should render this file.

DM_RENDERFILE - Syntax

This message is sent to the caller of [DrgDragFiles](#) to tell it to render a file.

```
param1
    PRENDERFILE  rndf          /* Pointer to a RENDERFILE structure. */

param2
    ULONG        ulReserved    /* Reserved value, should be 0. */
```

DM_RENDERFILE - Remarks

This message is sent when is specified in [DrgDragFiles](#). The receiver should perform the operation indicated by the TRUE field in the [RENDERFILE](#) structure, moving or copying *hstrSource* to *hstrTarget*.

When the operation is complete, a [DM_FILERENDERED](#) message should be sent to *hwndDragFiles* window.

The [RENDERFILE](#) structure is allocated temporarily for the receiver of this message. The receiver should make a copy if it needs to use the data in this structure after returning.

DM_RENDERFILE - Default Processing

The [WinDefWindowProc](#) function does not expect to receive this message and takes no action other than to return 0.

DM_RENDERFILE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

DM_RENDERPREPARE

DM_RENDERPREPARE Field - pDxfer

pDxfer ([PDRAGTRANSFER](#))
Pointer to a [DRAGTRANSFER](#) structure.

DM_RENDERPREPARE Field - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

DM_RENDERPREPARE Return Value - rc

rc (BOOL)

Success indicator.

TRUE

The message was processed by the recipient and it is ready to perform the rendering operation. The target of the drop sends a [DM_RENDER](#) message to request the rendering with a specific rendering mechanism and format.

FALSE

The message either was not processed by the recipient, or it is unprepared to perform the rendering. The *hwndItem* field in [DRAGITEM](#) may not be properly initialized, and therefore the target should not send a [DM_ENDCONVERSATION](#) message.

DM_RENDERPREPARE - Parameters

pDxfer (PDAGTRANSFER)

Pointer to a [DRAGTRANSFER](#) structure.

ulReserved (ULONG)

Reserved value, should be 0.

rc (BOOL)

Success indicator.

TRUE

The message was processed by the recipient and it is ready to perform the rendering operation. The target of the drop sends a [DM_RENDER](#) message to request the rendering with a specific rendering mechanism and format.

FALSE

The message either was not processed by the recipient, or it is unprepared to perform the rendering. The *hwndItem* field in [DRAGITEM](#) may not be properly initialized, and therefore the target should not send a [DM_ENDCONVERSATION](#) message.

DM_RENDERPREPARE - Syntax

This message tells a source to prepare for the rendering of an object.

```
param1
    PDRAGTRANSFER    pDxfer        /* Pointer to a DRAGTRANSFER structure. */

param2
    ULONG             ulReserved    /* Reserved value, should be 0. */
```

DM_RENDERPREPARE - Remarks

This message must be sent when DC_PREPARE is on in the [DRAGITEM](#) structure.

This message is used to allow the source to create an invisible window to handle the conversation required for the data transfer.

DM_RENDERPREPARE - Default Processing

The [WinDefWindowProc](#) function does not expect to receive this message and takes no action other than to return 0.

DM_RENDERPREPARE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

Dynamic Data Exchange Messages

This section describes the message part of the DDE protocol, which is a set of guidelines that allows two applications to share data freely between one another; not necessarily driven directly by user input.

Note: DDE operates between two specific applications, each of which must be aware of the other, and active.

[WinDdeInitiate](#), [WinDdePostMsg](#), and [WinDdeRespond](#) are the functions associated with these messages.

Dynamic Data Exchange Messages

This section describes the dynamic data exchange protocol actions on the following messages.

WM_DDE_ACK

WM_DDE_ACK Field - hwnd

hwnd ([HWND](#))
Window handle of the sender.

WM_DDE_ACK Field - pDdeStruct

pDdeStruct ([PDDESTRUCT](#))
DDE structure.

This points to a dynamic data exchange structure. See [DDESTRUCT](#).

The acknowledging application modifies the *fsStatus* field to return information about the status of the message received:

DDE_FACK	1 = request accepted 0 = request not accepted.
DDE_FBUSY	1 = busy 0 = not busy.
DDE_NOTPROCESSED DDE_FAPPSTATUS	Reserved for application-specific return codes The message was not understood and was ignored.

An application is expected to set DDE_FBUSY if it is unable to respond to the request at the time it is received. The DDE_FBUSY flag is defined only when DDE_FACK is 0.

offszItemName identifies the item for which the acknowledgment is being sent.

WM_DDE_ACK Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_DDE_ACK - Parameters

hwnd ([HWND](#))
Window handle of the sender.

pDdeStruct ([PDDESTRUCT](#))

DDE structure.

This points to a dynamic data exchange structure. See [DDESTRUCT](#).

The acknowledging application modifies the *fsStatus* field to return information about the status of the message received:

DDE_FACK	1 = request accepted 0 = request not accepted.
DDE_FBUSY	1 = busy 0 = not busy.
DDE_NOTPROCESSED DDE_FAPPSTATUS	Reserved for application-specific return codes The message was not understood and was ignored.

An application is expected to set DDE_FBUSY if it is unable to respond to the request at the time it is received. The DDE_FBUSY flag is defined only when DDE_FACK is 0.

offszItemName identifies the item for which the acknowledgment is being sent.

ulReserved (ULONG)

Reserved value, should be 0.

WM_DDE_ACK - Syntax

This message notifies an application of the receipt and processing of a [WM_DDE_EXECUTE](#), [WM_DDE_DATA](#), [WM_DDE_ADVISE](#), [WM_DDE_UNADVISE](#) or [WM_DDE_POKE](#) message, and in some cases, of a [WM_DDE_REQUEST](#) message.

This message is always posted.

```
param1
    HWND          hwnd          /* Window handle of the sender. */

param2
    PDDESTRUCT    pDdeStruct    /* DDE structure. */
```

WM_DDE_ACK - Default Processing

None.

WM_DDE_ACK - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Default Processing](#)

[Glossary](#)

WM_DDE_ADVISE

WM_DDE_ADVISE Field - hwnd

hwnd ([HWND](#))
Window handle of the sender.

WM_DDE_ADVISE Field - pDdeStruct

pDdeStruct ([PDDESTRUCT](#))
DDE structure.

This points to a dynamic data exchange structure. See [DDESTRUCT](#).

Flags in the *fsStatus* field are set as follows:

DDE_FACKREQ	If this bit is 1, the receiving (server) application is requested to send its WM_DDE_DATA messages with the acknowledgment-requested (DDE_FACKREQ) bit set. This offers a flow control technique, whereby the client application can avoid overload from incoming WM_DDE_DATA messages.
DDE_FNODATA	If this bit is 1, the server is requested to send its WM_DDE_DATA messages with a zero length data portion. These messages are alarms that tell the client the source data has changed. Upon receiving one of these alarms, the client can choose to call for the latest version of the data by issuing a WM_DDE_REQUEST message, or the client can choose to ignore the alarm. This is typically used when there is a significant resource cost associated with actually rendering and/or assimilating the data.

offszItemName identifies which data item is being requested.
usFormat is the preferred type of data of the client. It must be a registered DDE data format number.

WM_DDE_ADVISE Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_DDE_ADVISE - Parameters

hwnd ([HWND](#))
Window handle of the sender.

pDdeStruct ([PDDESTRUCT](#))
DDE structure.

This points to a dynamic data exchange structure. See [DDESTRUCT](#).

Flags in the *fsStatus* field are set as follows:

DDE_FACKREQ	If this bit is 1, the receiving (server) application is requested to send its WM_DDE_DATA messages with the acknowledgment-requested (DDE_FACKREQ) bit set. This offers a flow control technique, whereby the client application can avoid overload from incoming WM_DDE_DATA messages.
DDE_FNODATA	If this bit is 1, the server is requested to send its WM_DDE_DATA messages with a zero length data portion. These messages are alarms that tell the client the source data has changed. Upon receiving one of these alarms, the client can choose to call for the latest version of the data by issuing a WM_DDE_REQUEST message, or the client can choose to ignore the alarm. This is typically used when there is a significant resource cost associated with actually rendering and/or assimilating the data.

offszItemName identifies which data item is being requested.

usFormat is the preferred type of data of the client. It must be a registered DDE data format number.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_DDE_ADVISE - Syntax

This message (posted by a client application) requests the receiving application to supply an update for a data item whenever it changes.

This message is always posted.

```
param1
    HWND          hwnd          /* Window handle of the sender. */

param2
    PDDESTRUCT    pDdeStruct    /* DDE structure. */
```

WM_DDE_ADVISE - Remarks

The receiving application is expected to reply with a positive [WM_DDE_ACK](#) message if it can provide the requested data, or with a negative one if it can not.

WM_DDE_ADVISE - Default Processing

None.

WM_DDE_ADVISE - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

WM_DDE_DATA

WM_DDE_DATA Field - hwnd

hwnd ([HWND](#))
Window handle of the sender.

WM_DDE_DATA Field - pDdeStruct

pDdeStruct ([PDDESTRUCT](#))
DDE structure.

This points to a dynamic data exchange structure. See [DDESTRUCT](#).

Flags in the *fsStatus* field are set as follows:

DDE_FACKREQ	If this bit is 1, the receiving (client) application is expected to send a WM_DDE_ACK message after the memory object has been processed. If it is 0, the client application should not send a WM_DDE_ACK message.
DDE_FRESPONSE	If this bit is 1, this data is offered in response to a WM_DDE_REQUEST message. If it is 0, this data is offered in response to a WM_DDE_ADVISE message.

offszItemName identifies which data item is available.

offabData is the data. The format of the data is a registered DDE data format, identified by the *usFormat* field.

WM_DDE_DATA Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_DDE_DATA - Parameters

hwnd ([HWND](#))
Window handle of the sender.

pDdeStruct ([PDDESTRUCT](#))
DDE structure.

This points to a dynamic data exchange structure. See [DDESTRUCT](#).

Flags in the *fsStatus* field are set as follows:

DDE_FACKREQ	If this bit is 1, the receiving (client) application is expected to send a WM_DDE_ACK message after the memory object has been processed. If it is 0, the client application should not send a WM_DDE_ACK message.
DDE_FRESPONSE	If this bit is 1, this data is offered in response to a WM_DDE_REQUEST message. If it is 0, this data is offered in response to a WM_DDE_ADVISE message.

offszItemName identifies which data item is available.

offabData is the data. The format of the data is a registered DDE data format, identified by the *usFormat* field.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_DDE_DATA - Syntax

This message notifies a client application of the availability of data. It is always posted.

```
param1      HWND      hwnd      /* Window handle of the sender. */
param2      PDDESTRUCT pDdeStruct /* DDE structure. */
```

WM_DDE_DATA - Default Processing

None.

WM_DDE_DATA - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Default Processing](#)

[Glossary](#)

WM_DDE_EXECUTE

WM_DDE_EXECUTE Field - hwnd

hwnd ([HWND](#))

Window handle of the server.

WM_DDE_EXECUTE Field - pDdeStruct

pDdeStruct ([PDDESTRUCT](#))

DDE structure.

This points to a dynamic data exchange structure. See [DDESTRUCT](#).

offabData contains the commands to be executed.

WM_DDE_EXECUTE Return Value - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_DDE_EXECUTE - Parameters

hwnd ([HWND](#))

Window handle of the server.

pDdeStruct ([PDDESTRUCT](#))

DDE structure.

This points to a dynamic data exchange structure. See [DDESTRUCT](#).

offabData contains the commands to be executed.

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_DDE_EXECUTE - Syntax

This message posts a string to a server application to be processed as a series of commands. The server application is expected to post a [WM_DDE_ACK](#) message in response.

This message is always posted.

```
param1  
    HWND          hwnd          /* Window handle of the server. */  
  
param2  
    PDDESTRUCT    pDdeStruct    /* DDE structure. */
```

WM_DDE_EXECUTE - Remarks

The receiving application responds with a positive [WM_DDE_ACK](#) message if it can honor the request, otherwise it responds with a negative [WM_DDE_ACK](#) message.

The data portion of the [WM_DDE_ACK](#) message, in either case, must contain the original command string. The requesting application compares the command string in the acknowledgement to its original request to determine which request the acknowledgement is for.

The response is posted to the window handle specified in the first message parameter.

WM_DDE_EXECUTE - Default Processing

None.

WM_DDE_EXECUTE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)

WM_DDE_INITIATE

WM_DDE_INITIATE Field - hwnd

hwnd ([HWND](#))
Window handle of the sender.

WM_DDE_INITIATE Field - pData

pData ([PDDEINIT](#))
Pointer to initiation data.

This points to a [DDEINIT](#) structure. *pszAppName* is the name of the desired server application; if this is a zero-length string, any application can respond. *pszTopic* is the name of the desired topic; if this is a zero-length string, each responding application responds once for each topic that it can support.

WM_DDE_INITIATE Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WM_DDE_INITIATE - Parameters

hwnd ([HWND](#))
Window handle of the sender.

pData ([PDDEINIT](#))
Pointer to initiation data.

This points to a [DDEINIT](#) structure. *pszAppName* is the name of the desired server application; if this is a zero-length string, any application can respond. *pszTopic* is the name of the desired topic; if this is a zero-length string, each responding application responds once for each topic that it can support.

rc ([BOOL](#))

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WM_DDE_INITIATE - Syntax

This message is sent by an application to one or more other applications, to request initiation of a conversation.

This message is always sent.

```
param1
    HWND    hwnd    /* Window handle of the sender. */

param2
    PDDEINIT pData    /* Pointer to initiation data. */
```

WM_DDE_INITIATE - Remarks

Upon receiving this message, all applications with names matching the application name (where specified), that support the topic identified by the topic name, are expected to acknowledge.

A modal window, for example a message box, must not be invoked during the processing of this message.

WM_DDE_INITIATE - Default Processing

The default window procedure frees the segment referenced by *param2*.

WM_DDE_INITIATE - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

WM_DDE_INITIATEACK

WM_DDE_INITIATEACK Field - hwnd

hwnd ([HWND](#))
Window handle of the sender.

WM_DDE_INITIATEACK Field - pData

pData ([PDDEINIT](#))
Pointer to initiation data.

This points to a [DDEINIT](#) structure. *pszAppName* is the name of the responding server application; it must not be a zero-length string. *pszTopic* is the name of the topic that the server is willing to support; it must not be a zero-length string.

The [DDEINIT](#) structure must be in a shareable segment; it is the responsibility of the receiving window procedure to free this segment.

WM_DDE_INITIATEACK Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WM_DDE_INITIATEACK - Parameters

hwnd ([HWND](#))
Window handle of the sender.

pData ([PDDEINIT](#))
Pointer to initiation data.

This points to a [DDEINIT](#) structure. *pszAppName* is the name of the responding server application; it must not be a zero-length

string. *pszTopic* is the name of the topic that the server is willing to support; it must not be a zero-length string.

The [DDEINIT](#) structure must be in a shareable segment; it is the responsibility of the receiving window procedure to free this segment.

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WM_DDE_INITIATEACK - Syntax

This message is sent by a server application in response to a [WM_DDE_INITIATE](#) message, for each topic that the server application wishes to support.

```
param1
    HWND        hwnd    /* Window handle of the sender. */

param2
    PDDEINIT    pData    /* Pointer to initiation data. */
```

WM_DDE_INITIATEACK - Remarks

A modal window, such as a message box, must not be posted during the processing of this message.

WM_DDE_INITIATEACK - Default Processing

The default window procedure frees the segment referenced by *param2*.

WM_DDE_INITIATEACK - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

WM_DDE_POKE

WM_DDE_POKE Field - hwnd

hwnd ([HWND](#))
Window handle of the sender.

WM_DDE_POKE Field - pDdeStruct

pDdeStruct ([PDDESTRUCT](#))
DDE structure.

This points to a dynamic data exchange structure. See [DDESTRUCT](#).

offszItemName identifies the data item to the receiving application.

offabData is the data. The format of the data is a registered DDE data format, identified by the *usFormat* field.

WM_DDE_POKE Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_DDE_POKE - Parameters

hwnd ([HWND](#))
Window handle of the sender.

pDdeStruct ([PDDESTRUCT](#))
DDE structure.

This points to a dynamic data exchange structure. See [DDESTRUCT](#).

offszItemName identifies the data item to the receiving application.

offabData is the data. The format of the data is a registered DDE data format, identified by the *usFormat* field.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_DDE_POKE - Syntax

This message requests an application to accept an unsolicited data item. It is always posted.

```
param1      HWND      hwnd      /* Window handle of the sender. */  
param2      PDDESTRUCT pDdeStruct /* DDE structure. */
```

WM_DDE_POKE - Remarks

The receiving application is expected to reply with a positive [WM_DDE_ACK](#) message if it accepts the unsolicited data, or with a negative [WM_DDE_ACK](#) if it does not.

WM_DDE_POKE - Default Processing

None.

WM_DDE_POKE - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_DDE_REQUEST

WM_DDE_REQUEST Field - hwnd

hwnd ([HWND](#))
Window handle of the server.

WM_DDE_REQUEST Field - DdeStruct

DdeStruct ([PDDESTRUCT](#))

DDE structure.

This points to a dynamic data exchange structure. See [DDESTRUCT](#).

offszItemName identifies which data item is being requested.

usFormat identifies in which registered DDE data format the data item is to be rendered.

WM_DDE_REQUEST Return Value - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_DDE_REQUEST - Parameters

hwnd ([HWND](#))

Window handle of the server.

DdeStruct ([PDDESTRUCT](#))

DDE structure.

This points to a dynamic data exchange structure. See [DDESTRUCT](#).

offszItemName identifies which data item is being requested.

usFormat identifies in which registered DDE data format the data item is to be rendered.

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_DDE_REQUEST - Syntax

This message is posted from client to server, to request that the server provide a data item to the client.

This message is always posted.

```
param1  
    HWND          hwnd          /* Window handle of the server. */
```

```
param2
    PDDESTRUCT DdeStruct    /* DDE structure. */
```

WM_DDE_REQUEST - Remarks

The receiving application is expected to respond with a [WM_DDE_DATA](#) message, containing the requested data, if possible. Otherwise, it is expected to respond with a negative [WM_DDE_ACK](#) message.

WM_DDE_REQUEST - Default Processing

None.

WM_DDE_REQUEST - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_DDE_TERMINATE

WM_DDE_TERMINATE Field - hwnd

hwnd ([HWND](#))
Window handle of the sender.

WM_DDE_TERMINATE Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_DDE_TERMINATE Return Value - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

WM_DDE_TERMINATE - Parameters

hwnd (HWND)

Window handle of the sender.

ulReserved (ULONG)

Reserved value, should be 0.

ulReserved (ULONG)

Reserved value, should be 0.

WM_DDE_TERMINATE - Syntax

This message is posted by either application participating in a DDE conversation, to terminate that conversation.

This message is always posted.

```
param1
    HWND    hwnd        /* Window handle of the sender. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */
```

WM_DDE_TERMINATE - Remarks

Upon receiving this message, an application is expected to post a WM_DDE_TERMINATE message in response.

WM_DDE_TERMINATE - Default Processing

None.

WM_DDE_TERMINATE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_DDE_UNADVISE

WM_DDE_UNADVISE Field - hwnd

hwnd ([HWND](#))
Window handle of a sender.

WM_DDE_UNADVISE Field - DdeStruct

DdeStruct ([PDDESTRUCT](#))
DDE structure.

This points to a dynamic data exchange structure (see [DDESTRUCT](#)). *offset/ItemName* identifies which data update request is to be retracted. If this is a zero-length string, data update requests for all items are retracted.

WM_DDE_UNADVISE Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_DDE_UNADVISE - Parameters

hwnd ([HWND](#))

Window handle of a sender.

DdeStruct ([PDDESTRUCT](#))

DDE structure.

This points to a dynamic data exchange structure (see [DDESTRUCT](#)). *offs/ItemName* identifies which data update request is to be retracted. If this is a zero-length string, data update requests for all items are retracted.

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_DDE_UNADVISE - Syntax

This message is posted by a client application to a server application to indicate that the specified item should no longer be updated.

This message is always posted.

```
param1
    HWND      hwnd      /* Window handle of a sender. */

param2
    PDDESTRUCT DdeStruct /* DDE structure. */
```

WM_DDE_UNADVISE - Remarks

The receiving application is expected to reply with a positive [WM_DDE_ACK](#) message if it can honor the request, or a negative one if it cannot.

WM_DDE_UNADVISE - Default Processing

None.

WM_DDE_UNADVISE - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

Entry Field Control Window Processing

This system-provided window procedure processes the actions on an entry field control (WC_ENTRYFIELD).

Purpose

An entry field control is a rectangular window that displays a single line of text that the operator can edit. When it has the focus, the cursor marks the current **insertion** or **replacement** point.

When working with entry fields, the [WM_CONTROL](#) message is of major concern. An entry-field control communicates with its owner by sending [WM_CONTROL](#) messages. It contains a notification code in MP1 and a handle to the current entry field in MP2. The return value for [WM_CONTROL](#) is 0. Notification codes are denoted by an EN prefix.

For entry field control data see, [ENTRYFDATA](#).

Entry Field Control Styles

These entry field control styles are available:

ES_LEFT	The text in the control is left-justified. This is the default style if neither ES_RIGHT nor ES_CENTER is specified.
ES_RIGHT	The text in the control is right-justified.
ES_CENTER	The text in the control is centered.
ES_AUTOSIZE	The text will be sized to make sure the contents fit.
ES_AUTOSCROLL	If the user tries to move off the end of a line, the control automatically scrolls one-third the width of the window in the appropriate direction.
ES_MARGIN	<p>This style can be used to cause a border to be drawn around the control, with a margin around the editable text. The margin is half a character-width wide and half a character-height high.</p> <p>When an entry field control with this style is positioned, it adjusts the position so that the text is placed at the position specified. This position differs from the original position by the width of the border and the margin.</p>
ES_READONLY	<p>This style causes a single line entry field to be created in read only state.</p> <p>When an entry field is in read only state, characters do not get inserted into the text. However the insertion interface is still functional.</p> <p>The entry field read only state can be altered by use of the EM_SETREADONLY message.</p>
ES_UNREADABLE	This style causes the text to be displayed as an asterisk for each character. It can be used for passwords.
ES_COMMAND	<p>This style identifies the entry field as a command entry field. This information is used by the Help Manager to provide command help if the end user requests help for this field.</p> <p>Not more than one entry field on each dialog should be given this style.</p>
ES_AUTOTAB	<p>This style indicates that when the field is filled by adding a character to the end of the entry field text, the effect of a tab key will be generated. Inserting or replacing a character in the middle of the text, however, does not result in an autotab.</p> <p>This style is recommended for use with fixed-length, non-scrollable fields that are filled completely. The maximum length of the entry field text is held in the control data, see ENTRYFDATA.</p>

These entry field controls are intended for countries that use a double-byte character encoding scheme:

ES_SBCS	<p>The text is purely single-byte.</p> <p>If the number of characters entered exceeds EM_SETTEXTLIMIT, or a DBCS character is entered, the alarm sounds and the last character entered is ignored.</p>
ES_DBCS	<p>The text is purely double byte.</p> <p>If the number of bytes in the entry field exceeds EM_SETTEXTLIMIT, or an SBCS character is entered, the alarm sounds and the last character entered is ignored.</p>
ES_ANY	<p>The text is a mixture of SBCS and DBCS characters.</p> <p>If the number of bytes in the input field exceeds EM_SETTEXTLIMIT, the alarm sounds and the last character entered is ignored.</p> <p>ES_ANY is the default.</p> <p>Note: If the queue code page is an ASCII code page and the data in the entry field is to be converted to an EBCDIC code page, there is a possibility that shift-in and shift-out characters introduced by the conversion process can cause the converted data to overrun the target field. Coding ES_MIXED protects the target field from overrun in this situation.</p>
ES_MIXED	<p>The text is a mixture of SBCS and DBCS characters which may subsequently be converted from an ASCII DBCS code page to an EBCDIC DBCS code page with a consequent possible increase in the length of the data.</p> <p>If</p> $DBCSchars*2 + SBCSchars + N > EM_SETTEXTLIMIT$ <p>where N starts at 0 and is incremented whenever the string goes from SBCS to DBCS or DBCS to SBCS, the alarm sounds and the last character entered is ignored.</p> <p>Note: For every conversion from SBCS to DBCS there must be a corresponding return to SBCS (N must be an even number).</p>

Default Colors

The following system colors are used when the system draws button controls:

```

SYSCLR_ENTRYFIELD
SYSCLR_BUTTONDARK
SYSCLR_BUTTONLIGHT
SYSCLR_OUTPUTTEXT
SYSCLR_WINDOWTEXT
SYSCLR_HIGHLIGHTFOREGROUND
SYSCLR_HIGHLIGHTBACKGROUND

```

Some of these defaults can be replaced by using the following presentation parameters in the application resource script file or source code:

```

PP_FOREGROUNDCOLOR
PP_DISABLEDFOREGROUNDCOLOR
PP_HIGHLIGHTFOREGROUNDCOLOR
PP_FONTNAMESIZE

```

Entry Field Control Notification Messages

This message is initiated by the entry field control window to notify its owner of significant events.

WM_CONTROL (in Entry Fields)

WM_CONTROL (in Entry Fields) Field - id

id ([USHORT](#))
Control window identity.

WM_CONTROL (in Entry Fields) Field - usnotifycode

usnotifycode ([USHORT](#))
Notify code.

- EN_CHANGE** The content of the entry field control has changed, and the change has been displayed on the screen.
- EN_KILLFOCUS** The entry field control is losing the focus.
- EN_MEMERROR** The entry field control cannot allocate the storage necessary to accommodate window text of the length implied by the [EM_SETTEXTLIMIT](#) message.
- EN_OVERFLOW** The entry field control cannot insert more text than the current text limit. The text limit may be changed with the [EM_SETTEXTLIMIT](#) message.
- If the recipient of this message returns TRUE, then the entry field control retries the operation, otherwise it terminates the operation.
- EN_SCROLL** The entry field control is about to scroll horizontally. This can happen in these circumstances:
- The application has issued a [WinScrollWindow](#) call
 - The content of the entry field control has changed
 - The caret has moved
 - The entry field control must scroll to show the caret position.
- EN_SETFOCUS** The entry field control is receiving the focus.
-

WM_CONTROL (in Entry Fields) Field - hwndcontrolspect

hwndcontrolspect ([HWND](#))
Entry field control window handle.

WM_CONTROL (in Entry Fields) Return Value - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_CONTROL (in Entry Fields) - Parameters

id ([USHORT](#))

Control window identity.

usnotifycode ([USHORT](#))

Notify code.

EN_CHANGE

The content of the entry field control has changed, and the change has been displayed on the screen.

EN_KILLFOCUS

The entry field control is losing the focus.

EN_MEMERROR

The entry field control cannot allocate the storage necessary to accommodate window text of the length implied by the [EM_SETTEXTLIMIT](#) message.

EN_OVERFLOW

The entry field control cannot insert more text than the current text limit. The text limit may be changed with the [EM_SETTEXTLIMIT](#) message.

If the recipient of this message returns TRUE, then the entry field control retries the operation, otherwise it terminates the operation.

EN_SCROLL

The entry field control is about to scroll horizontally. This can happen in these circumstances:

- The application has issued a [WinScrollWindow](#) call
- The content of the entry field control has changed
- The caret has moved
- The entry field control must scroll to show the caret position.

EN_SETFOCUS

The entry field control is receiving the focus.

hwndcontrolspect ([HWND](#))

Entry field control window handle.

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_CONTROL (in Entry Fields) - Syntax

For the cause of this message, see [WM_CONTROL](#).

```
param1
    USHORT id          /* Control window identity. */
    USHORT usnotifycode /* Notify code. */

param2
    HWND hwndcontrolspe /* Entry field control window handle. */
```

WM_CONTROL (in Entry Fields) - Remarks

The entry field control window procedure generates this message and sends it to its owner, informing the owner of the event.

WM_CONTROL (in Entry Fields) - Default Processing

The default window procedure takes no action on this message, other than to set *uReserved* to 0.

WM_CONTROL (in Entry Fields) - Related Messages

Related Messages

- [WM_CONTROL](#)

WM_CONTROL (in Entry Fields) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

Entry Field Control Window Messages

This section describes the entry field control window procedure actions on receiving these messages:

EM_CLEAR

EM_CLEAR Field - ulReserve

ulReserve (ULONG)
Reserved value, should be 0.

EM_CLEAR Field - ulReserve

ulReserve (ULONG)
Reserved value, should be 0.

EM_CLEAR Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

EM_CLEAR - Parameters

ulReserve (ULONG)
Reserved value, should be 0.

ulReserve (ULONG)
Reserved value, should be 0.

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

EM_CLEAR - Syntax

This message deletes the text that forms the current selection.

```
param1
    ULONG  ulReserved /* Reserved value, should be 0. */

param2
    ULONG  ulReserved /* Reserved value, should be 0. */
```

EM_CLEAR - Remarks

The entry field control window procedure responds to this message by deleting the text that forms the current selection and setting *usmaxsel* equal to *usminsel*.

EM_CLEAR - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

EM_CLEAR - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

EM_COPY

EM_COPY Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

EM_COPY Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

EM_COPY Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

EM_COPY - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

EM_COPY - Syntax

This message copies the current selection to the clipboard.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

EM_COPY - Remarks

The entry field control window procedure responds to this message by copying the text that forms the current selection to the clipboard in CF_TEXT format.

EM_COPY - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

EM_COPY - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

EM_CUT

EM_CUT Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

EM_CUT Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

EM_CUT Return Value - rc

rc (BOOL)	Success indicator.
TRUE	Successful completion
FALSE	Error occurred.

EM_CUT - Parameters

ulReserved (ULONG)	Reserved value, should be 0.
ulReserved (ULONG)	Reserved value, should be 0.

rc (BOOL)	Success indicator.
TRUE	Successful completion
FALSE	Error occurred.

EM_CUT - Syntax

This message copies the text that forms the current selection to the clipboard, and then deletes it from the entry field control.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */
param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

EM_CUT - Remarks

The entry field control window procedure responds to this message by copying the text that forms the current selection to the clipboard in CF_TEXT format, and then deleting it from the entry field control and setting *usmaxsel* equal to *usminsel*.

This message is the combination of a [EM_COPY](#) message followed by a [EM_CLEAR](#) message.

EM_CUT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

EM_CUT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

EM_PASTE

EM_PASTE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

EM_PASTE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

EM_PASTE Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	

Error occurred.

For example, if the text to be inserted does not fit in the entry field control without overflowing the text limit set by the [EM_SETTEXTLIMIT](#) message, in which instance no text is inserted.

EM_PASTE - Parameters

ulReserved ([ULONG](#))

Reserved value, should be 0.

ulReserved ([ULONG](#))

Reserved value, should be 0.

rc ([BOOL](#))

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

For example, if the text to be inserted does not fit in the entry field control without overflowing the text limit set by the [EM_SETTEXTLIMIT](#) message, in which instance no text is inserted.

EM_PASTE - Syntax

This message replaces the text that forms the current selection with text from the clipboard.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */
param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

EM_PASTE - Remarks

The entry field control window procedure responds to this message by replacing the text that forms the current selection with text from the clipboard, if the data is in CF_TEXT format.

Only characters from the clipboard up to the first carriage return are used in the replacement.

EM_PASTE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the

default value of FALSE.

EM_PASTE - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

EM_QUERYCHANGED

EM_QUERYCHANGED Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

EM_QUERYCHANGED Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

EM_QUERYCHANGED Return Value - rc

- rc** (BOOL)
Changed indicator.
- | | |
|-------|--|
| TRUE | The text in the entry field control has been changed since the last time it received this message or a WM_QUERYWINDOWPARAMS message. |
| FALSE | All other situations. |

EM_QUERYCHANGED - Parameters

ulReserved (ULONG)

Reserved value, should be 0.

ulReserved (ULONG)

Reserved value, should be 0.

rc (BOOL)

Changed indicator.

TRUE

The text in the entry field control has been changed since the last time it received this message or a [WM_QUERYWINDOWPARAMS](#) message.

FALSE

All other situations.

EM_QUERYCHANGED - Syntax

This message enquires if the text of the entry field control has been changed since the last enquiry.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */
param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

EM_QUERYCHANGED - Remarks

The entry field control window procedure responds to this message by setting *rc* to indicate whether the text of the entry field has been changed since the last time either this message or a [WM_QUERYWINDOWPARAMS \(in Entry Fields\)](#) message has been received.

EM_QUERYCHANGED - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

EM_QUERYCHANGED - Topics

Select an item:

EM_QUERYFIRSTCHAR

EM_QUERYFIRSTCHAR Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

EM_QUERYFIRSTCHAR Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

EM_QUERYFIRSTCHAR Return Value - sOffset

sOffset ([SHORT](#))
Zero-based offset.

EM_QUERYFIRSTCHAR - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

sOffset ([SHORT](#))
Zero-based offset.

EM_QUERYFIRSTCHAR - Syntax

This message returns the zero-based offset of the first character visible at the left edge of an entry-field control.

```
param1
    ULONG    ulReserved /* Reserved value, should be 0. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */
```

EM_QUERYFIRSTCHAR - Remarks

The entry field control window procedure responds to this message by returning the zero-based offset into the text that corresponds to the first character displayed in the entry field control.

EM_QUERYFIRSTCHAR - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sOffset* to the default value of 0.

EM_QUERYFIRSTCHAR - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

EM_QUERYREADONLY

EM_QUERYREADONLY Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

EM_QUERYREADONLY Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

EM_QUERYREADONLY Return Value - rc

rc (BOOL)
Read only state indicator.

TRUE	Read only state is enabled.
FALSE	Read only state is disabled.

EM_QUERYREADONLY - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

rc (BOOL)
Read only state indicator.

TRUE	Read only state is enabled.
FALSE	Read only state is disabled.

EM_QUERYREADONLY - Syntax

This message returns the read only state of an entry field control.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */
param2
```


`ULONG ulReserved /* Reserved value, should be 0. */`

EM_QUERYREADONLY - Remarks

The entry field control window procedure responds to this message by returning the read only state of the entry field control.

EM_QUERYREADONLY - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set `rc` to the default value of FALSE.

EM_QUERYREADONLY - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

EM_QUERYSEL

EM_QUERYSEL Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

EM_QUERYSEL Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

EM_QUERYSEL Field - sMinSel

sMinSel ([SHORT](#))

Offset of the first character in the selection.

EM_QUERYSEL Field - sMaxSel

sMaxSel ([SHORT](#))

Offset of the first character after the selection.

EM_QUERYSEL - Parameters

ulReserved ([ULONG](#))

Reserved value, should be 0.

ulReserved ([ULONG](#))

Reserved value, should be 0.

sMinSel ([SHORT](#))

Offset of the first character in the selection.

sMaxSel ([SHORT](#))

Offset of the first character after the selection.

EM_QUERYSEL - Syntax

This message gets the zero-based offsets of the bounds of the text that forms the current selection.

```
param1
    ULONG  ulReserved  /* Reserved value, should be 0. */

param2
    ULONG  ulReserved  /* Reserved value, should be 0. */

returns
    SHORT  sMinSel      /* Offset of the first character in the selection. */
    SHORT  sMaxSel      /* Offset of the first character after the selection. */
```

EM_QUERYSEL - Remarks

The entry field control window procedure responds to this message by returning the zero-based offsets of the bounds of the text that forms the current selection.

EM_QUERYSEL - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sMinSel* to the default value of 0, which is equivalent to setting both *sMinSel* and *sMaxSel* to 0.

EM_QUERYSEL - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

EM_SETFIRSTCHAR

EM_SETFIRSTCHAR Field - sOffset

sOffset ([SHORT](#))
Zero-based offset of the first character to be displayed.

EM_SETFIRSTCHAR Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

EM_SETFIRSTCHAR Return Value - rc

rc (BOOL)

Success indicator.

TRUE

Successful completion

FALSE

Error occurred. For example, because *sOffset* is not valid.

EM_SETFIRSTCHAR - Parameters

sOffset (SHORT)

Zero-based offset of the first character to be displayed.

ulReserved (ULONG)

Reserved value, should be 0.

rc (BOOL)

Success indicator.

TRUE

Successful completion

FALSE

Error occurred. For example, because *sOffset* is not valid.

EM_SETFIRSTCHAR - Syntax

This message specifies the offset of the character to be displayed in the first position of the entry field control.

```
param1
    SHORT  sOffset    /* Zero-based offset of the first character to be displayed. */

param2
    ULONG  ulReserved /* Reserved value, should be 0. */
```

EM_SETFIRSTCHAR - Remarks

The entry field control window procedure responds to this message by setting the text displayed in the edit control so that the first character displayed on the left of the window has the zero-based index specified by *sOffset*.

An EN_SCROLL notification message occurs, if the entry field control scrolls. This message returns FALSE if the edit control does not have the ES_AUTOSCROLL style or it is center of right justified.

EM_SETFIRSTCHAR - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the

default value of FALSE.

EM_SETFIRSTCHAR - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

EM_SETINSERTMODE

EM_SETINSERTMODE Field - usInsert

usInsert (USHORT)	
Insert mode indicator.	
TRUE	
	Enable insert mode.
FALSE	
	Enable overtype mode.

EM_SETINSERTMODE Field - ulReserved

ulReserved (ULONG)	
Reserved value, should be 0.	

EM_SETINSERTMODE Return Value - rc

rc (BOOL)	
Previous insert mode indicator.	
TRUE	
	Insert mode was previously enabled.
FALSE	
	Overtyping mode was previously enabled.

EM_SETINSERTMODE - Parameters

usInsert (USHORT)

Insert mode indicator.

TRUE

Enable insert mode.

FALSE

Enable oertype mode.

ulReserved (ULONG)

Reserved value, should be 0.

rc (BOOL)

Previous insert mode indicator.

TRUE

Insert mode was previously enabled.

FALSE

Overtime mode was previously enabled.

EM_SETINSERTMODE - Syntax

This message sets the insert mode of an entry field.

```
param1
    USHORT  usInsert    /* Insert mode indicator. */

param2
    ULONG   ulReserved  /* Reserved value, should be 0. */
```

EM_SETINSERTMODE - Remarks

The entry field control window procedure responds to this message by setting the insert mode of the entry field, updating the SV_INSERTMODE system constant and redrawing the entry field.

EM_SETINSERTMODE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

EM_SETINSERTMODE - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

EM_SETREADONLY

EM_SETREADONLY Field - usReadOnly

usReadOnly ([USHORT](#))
Read only state indicator.

TRUE	Enable read only state
FALSE	Disable read only state.

EM_SETREADONLY Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

EM_SETREADONLY Return Value - rc

rc ([BOOL](#))
Previous read only state indicator.

TRUE	Read only state was previously enabled.
FALSE	Read only state was previously disabled.

EM_SETREADONLY - Parameters

usReadOnly ([USHORT](#))
Read only state indicator.

TRUE	Enable read only state
FALSE	Disable read only state.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Previous read only state indicator.

TRUE	Read only state was previously enabled.
FALSE	Read only state was previously disabled.

EM_SETREADONLY - Syntax

This message sets the read only state of an entry field control.

```
param1
    USHORT  usReadOnly /* Read only state indicator. */

param2
    ULONG   ulReserved /* Reserved value, should be 0. */
```

EM_SETREADONLY - Remarks

The entry field control window procedure responds to this message by setting the read only state of the entry field control.

EM_SETREADONLY - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

EM_SETREADONLY - Topics

Select an item:

EM_SETSEL

EM_SETSEL Field - usminsel

usminsel ([USHORT](#))
Offset of the first character in the selection.

EM_SETSEL Field - usmaxsel

usmaxsel ([USHORT](#))
Offset of the first character after the selection.

EM_SETSEL Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

EM_SETSEL Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

EM_SETSEL - Parameters

usminsel ([USHORT](#))

Offset of the first character in the selection.

usmaxsel ([USHORT](#))

Offset of the first character after the selection.

ulReserved ([ULONG](#))

Reserved value, should be 0.

rc ([BOOL](#))

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

EM_SETSEL - Syntax

This message sets the zero-based offsets of the bounds of the text that forms the current selection.

```
param1
    USHORT  usminsel    /* Offset of the first character in the selection. */
    USHORT  usmaxsel    /* Offset of the first character after the selection. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */
```

EM_SETSEL - Remarks

The entry field control window procedure responds to this message by setting the zero-based offsets of the bounds of the text that forms the current selection.

If *usminsel* equals *usmaxsel*, the current selection becomes an insertion point.

If *usminsel* equals 0 and *usmaxsel* is equal to or greater than the text limit set by the [EM_SETTEXTLIMIT](#) message, the entire text is selected. Selected text is displayed in reverse color.

EM_SETSEL - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

EM_SETSEL - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

EM_SETTEXTLIMIT

EM_SETTEXTLIMIT Field - sTextLimit

sTextLimit ([SHORT](#))
Maximum number of characters in the entry field control.

EM_SETTEXTLIMIT Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

EM_SETTEXTLIMIT Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred. For example, because not enough storage can be allocated.

EM_SETTEXTLIMIT - Parameters

sTextLimit ([SHORT](#))
Maximum number of characters in the entry field control.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred. For example, because not enough storage can be allocated.

EM_SETTEXTLIMIT - Syntax

This message sets the maximum number of bytes that an entry field control can contain.

```
param1
    SHORT sTextLimit /* Maximum number of characters in the entry field control. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

EM_SETTEXTLIMIT - Remarks

The entry field control window procedure responds to this message by setting the maximum number of characters that can be contained.

This message is intended only to limit the length of lines that result from the user interacting with the entry field control. It also limits the length of text that can result from sending a [EM_PASTE](#) or [WM_SETWINDOWPARAMS](#) message.

EM_SETTEXTLIMIT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

EM_SETTEXTLIMIT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)

WM_CHAR (in Entry Fields)

WM_CHAR (in Entry Fields) - Syntax

For the cause of this message, see [WM_CHAR](#).

For a description of the parameters, see [WM_CHAR](#).

WM_CHAR (in Entry Fields) - Remarks

The entry field control window procedure responds to this message by sending it to its owner if it has not processed the keystroke. This is the most common means by which the input focus is switched around the various controls in a dialog box.

Unlike other controls, the *usvk* field of the message [WM_CHAR](#) takes precedence over other fields only when the Shift key is pressed.

If this message contains a valid *usck* field of the message [WM_CHAR](#), that character is entered into the text in insert or overwrite mode.

The keystrokes processed by an entry field control are:

Left arrow	Move the cursor one character to the left.
Right arrow	Move the cursor one character to the right.
Shift+Left arrow	Extend the selection by one character to the left.
Shift+Right arrow	Extend the selection by one character to the right.
Home	Move the cursor to the beginning of the text.
End	Move the cursor to the end of the text.
Backspace	Delete the character to the left of the cursor.
Delete	When the selection is an insertion point, delete the character to the right of the cursor, otherwise delete the current selection, but do not put it in the clipboard.
Shift+Del	Cut the current selection to the clipboard.
Shift+Ins	Replace the current selection with the text contents from the clipboard.
Ctrl+Del	Delete to the end of the field.
Ctrl+Ins	Copy the current selection to the clipboard.

If the control contains more text than can be shown, the actions defined above that move the cursor cause the text to be scrolled. The amount of scrolling varies from key to key, and the position of the text within the control varies for the same cursor position.

WM_CHAR (in Entry Fields) - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message other than to set *rc* to FALSE.

WM_CHAR (in Entry Fields) - Related Messages

Related Messages

- [WM_CHAR](#)
-

WM_CHAR (in Entry Fields) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_QUERYCONVERTPOS (in Entry Fields)

WM_QUERYCONVERTPOS (in Entry Fields) - Syntax

For the cause of this message, see [WM_QUERYCONVERTPOS](#).

For a description of the parameters, see [WM_QUERYCONVERTPOS](#).

WM_QUERYCONVERTPOS (in Entry Fields) - Remarks

The entry field control window procedure updates *pCursorPos* to the position of the cursor and returns QCP_CONVERT.

WM_QUERYCONVERTPOS (in Entry Fields) - Default Processing

For the default window procedure processing of this message see [WM_QUERYCONVERTPOS](#).

WM_QUERYCONVERTPOS (in Entry Fields) - Related Messages

Related Messages

- [WM_QUERYCONVERTPOS](#)

WM_QUERYCONVERTPOS (in Entry Fields) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_QUERYWINDOWPARAMS (in Entry Fields)

WM_QUERYWINDOWPARAMS (in Entry Fields) - Syntax

This message occurs when an application queries the entry field control window parameters.

For a description of the parameters, see [WM_QUERYWINDOWPARAMS](#).

WM_QUERYWINDOWPARAMS (in Entry Fields) - Remarks

The entry field control window procedure responds to this message by returning the window parameters indicated by the *fsStatus* parameter of the [WNDPARAMS](#) data structure identified by the *pwndparams* parameter.

WM_QUERYWINDOWPARAMS (in Entry Fields) - Default Processing

The default window procedure sets the *cchText*, *cbPresParams*, and *cbCtlData* parameters of the [WNDPARAMS](#) data structure, identified by *pwndparams*, to 0 and sets *rc* to FALSE.

WM_QUERYWINDOWPARAMS (in Entry Fields) - Related Messages

Related Messages

- [WM_QUERYWINDOWPARAMS](#)

WM_QUERYWINDOWPARAMS (in Entry Fields) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_SETWINDOWPARAMS (in Entry Fields)

WM_SETWINDOWPARAMS (in Entry Fields) - Syntax

This message occurs when an application sets or changes the entry field control window parameters.

For a description of the parameters, see [WM_SETWINDOWPARAMS](#).

WM_SETWINDOWPARAMS (in Entry Fields) - Remarks

The entry field control window procedure responds to this message by setting the window parameters indicated by the *fsStatus* parameter of the [WNDPARAMS](#) data structure, identified by the *pwndparams* parameter.

WM_SETWINDOWPARAMS (in Entry Fields) - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_SETWINDOWPARAMS (in Entry Fields) - Related Messages

Related Messages

- [WM_SETWINDOWPARAMS](#)

WM_SETWINDOWPARAMS (in Entry Fields) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

Frame Control Window Processing

This system-provided window procedure processes the actions on a frame window (WC_FRAME). The frame control window procedure sends all messages not processed to FID_CLIENT and sets **reply** to 0.

- For a description of the frame creation flags and the frame control styles, see [Frame Creation Flags](#) and [Frame Control Styles](#).
- For frame control data, see [FRAMECDATA](#).

Purpose

The window that contains all of the parts listed below is called the *frame window*. Each of the parts that make up a window, such as the title bar and menu, are separate child windows of the frame window. All of these child windows, except the client window (FID_CLIENT), are called *frame controls*.

FID_CLIENT is not a frame control, it is an instance of a window class implemented by the application.

The frame window and all of the frame controls are implemented with system-provided preregistered window classes.

The frame window holds together all of the frame controls and FID_CLIENT that make up an application window. The frame window is responsible for arranging the frame controls and the FID_CLIENT as the frame window is sized and moved. It is also responsible for routing specific messages to its frame controls and the FID_CLIENT.

Each of the frame controls and FID_CLIENT are known to the frame window by a system-provided window-identifier value as listed below:

FID_CLIENT Client window
FID_HORZSCROLL Horizontal scroll bar
FID_MENU Application menu
FID_MINMAX Minimize/Maximize box
FID_SYSMENU System menu
FID_TITLEBAR Title bar
FID_VERTSCROLL Vertical scroll bar.

For correct operation, only one window per frame must be defined with each of the above FID_* values.

Frame Creation Flags

These frame creation flags are available:

FCF_TITLEBAR	Title bar.
FCF_SYSMENU	System menu.
FCF_MENU	Application menu.
FCF_MINMAX	Minimize and Maximize buttons.
FCF_MINBUTTON	Minimize button.
FCF_MAXBUTTON	Maximize button.
FCF_VERTSCROLL	Vertical scroll bar.
FCF_HORZSCROLL	Horizontal scroll bar.
FCF_SIZEBORDER	Sizing border.
FCF_BORDER	Window is drawn with a thin border.
FCF_DLGBORDER	Window is drawn with a standard dialog border.
FCF_ACCELTABLE	Causes an accelerator table to be loaded, for this frame window, from the resource file identified on the WinCreateStdWindow function.
FCF_ICON	<p>Window is created with an icon associated with it that is used to represent the window when it is minimized.</p> <p>If present, the parameter of the WinCreateStdWindow function must be the identity of an icon. This icon is loaded and associated with the window. When the window is minimized, the icon is shown if the screen is capable of showing it. When the window is destroyed, the icon is also destroyed.</p>
FCF_SHELLPOSITION	The window is created with a size and position determined by the shell, rather than explicitly by the application.
FCF_SYSMODAL	The frame window is System Modal.
FCF_NOBYTEALIGN	<p>When this flag is not set, the frame window is adjusted so that window operations, such as moving, can be performed in an optimized manner. For example, some displays can move a window more quickly if the movement is by a multiple of eight pels.</p> <p>If this flag is set, such optimizations are not performed and size and position values are honored.</p>
FCF_TASKLIST	<p>When this flag is set, the program title is added to the front of the frame window text, the resulting string is used as the window title and is also entered on the task list.</p> <p>In this context, the program title is the text string used by the Desktop Manager to identify the program, or the text string specified as a parameter in the START command. If neither string has been defined, the filename and extension of the .EXE file are used as the program title.</p> <p>Note that a WinSetWindowText will not change the entry in the switch list, a WinChangeSwitchEntry must be done to affect this.</p>
FCF_NOMOVEWITHOWNER	The window should not be moved when its owner is moved.
FCF_STANDARD	<p>Same as (FCF_TITLEBAR FCF_SYSMENU FCF_MINBUTTON FCF_MAXBUTTON FCF_SIZEBORDER FCF_ICON FCF_MENU FCF_ACCELTABLE FCF_SHELLPOSITION FCF_TASKLIST).</p> <p>This value is assumed if any Frame Window is created with no Control Data.</p>
FCF_SCREENALIGN	See FS_SCREENALIGN.

FCF_MOUSEALIGN	See FS_MOUSEALIGN.
FCF_AUTOICON	Performance optimization. When repainting iconized frames, the system will redraw the icon and will not send a WM_PAINT message to the application.
FCF_HIDEBUTTON	Hide button.
FCF_HIDEMAX	Hide and maximize buttons.

Frame Control Styles

These frame control styles are available. Frame styles may only be used when the frame is created from a dialog template.

FS_SCREENALIGN	The coordinates specifying the location of the dialog box are relative to the top left corner of the screen, rather than being relative to the owner window's origin.
FS_MOUSEALIGN	The coordinates specifying the location of the dialog box are relative to the position of the pointing device pointer at the time the window was created. The operating system tries to keep the dialog box on the screen, if possible.
FS_SIZEBORDER	See FCF_SIZEBORDER.
FS_BORDER	See FCF_BORDER.
FS_DLGBORDER	See FCF_DLGBORDER.
FS_SYSMODAL	See FCF_SYSMODAL.
FS_NOBYTEALIGN	See FCF_NOBYTEALIGN.
FS_TASKLIST	See FCF_TASKLIST.
FS_NOMOVEWITHOWNER	See FCF_NOMOVEWITHOWNER.
FS_AUTOICON	See FCF_AUTOICON.

Default Colors

The following system colors are used when the system draws frame controls:

- `SYSCLR_DIALOGBACKGROUND`
- `SYSCLR_ACTIVETITLE`
- `SYSCLR_INACTIVETITLE`
- `SYSCLR_APPWORKSPACE`
- `SYSCLR_ACTIVEBORDER`
- `SYSCLR_WINDOW`
- `SYSCLR_SHADOW`
- `SYSCLR_WINDOWFRAME`
- `SYSCLR_FIRST`.

Some of these defaults can be replaced by using the following presentation parameters in the application resource script file or source code:

```
PP_BACKGROUNDCOLOR
PP_SHADOW
PP_FOREGROUNDCOLOR
PP_BORDERCOLOR
PP_DISABLEDBACKGROUNDCOLOR.
```

Frame Control Notification Messages

These messages are initiated by the frame control window to notify the FID_CLIENT window.

WM_MINMAXFRAME (in Frame Controls)

WM_MINMAXFRAME (in Frame Controls) - Syntax

For the cause of this message, see [WM_MINMAXFRAME](#).

For a description of the parameters, see [WM_MINMAXFRAME](#).

WM_MINMAXFRAME (in Frame Controls) - Remarks

The window words are initialized before this message is sent. The window state has not been changed when this message is sent, and so the [WinQueryWindowPos](#) function can be used.

This message is sent by default to the FID_CLIENT window.

The system default actions, if FALSE is returned to this message, are based on the operation specified by the *pswp* parameter.

These actions affect the status of the frame window, and the title button windows and system menu windows contained within it, as follows:

- Window is maximized from a minimized state.
 - Title button windows:

The RESTORE button window is replaced by a MIN button window and the MAX button window is replaced by a RESTORE button window.
 - System menu window:

The MINIMIZE menu entry is enabled and the MAXIMIZE menu entry is disabled.
 - Other changes:

The frame window has the WS_MAXIMIZED style bit set and the WS_MINIMIZED style bit reset. Also the MS_VERTICALFLIP style bit of the system menu window is reset.
- Window is restored from a minimized state.
 - Title button windows:

The RESTORE button window is replaced by a MIN button window (the MAX button window is unaltered).
 - System menu window:

The MINIMIZE menu entry is enabled, the RESTORE menu entry is disabled and the SIZE menu entry is enabled.
 - Other changes:

The frame window has the WS_MINIMIZED style bit and the MS_VERTICALFLIP style bit of the system menu window reset.

- Window is minimized from a maximized state.
 - Title button windows:

The RESTORE button window is replaced by a MAX button window and the MIN button window is replaced by a RESTORE button window.
 - System menu window:

The MAXIMIZE menu entry is enabled and the MINIMIZE menu entry is disabled.
 - Other changes:

The frame window has the WS_MINIMIZED style bit set and the WS_MAXIMIZED style bit reset. Also the MS_VERTICALFLIP style bit of the system menu window is set.
- Window is restored from a maximized state.
 - Title button windows:

The RESTORE button window is replaced by a MAX button window (the MIN button window is unaltered).
 - System menu window:

The MAXIMIZE menu entry is enabled, the RESTORE menu entry is disabled and the SIZE menu entry is enabled.
 - Other changes:

The frame window has the WS_MAXIMIZED style bit reset.
- Window is minimized from a restored state.
 - Title-button windows:

The MIN button window is replaced by a RESTORE button window (the MAX button window is unaltered).
 - System menu window:

The RESTORE menu entry is enabled, the MINIMIZE menu entry is disabled and the SIZE menu entry is disabled.
 - Other changes:

The frame window has the WS_MINIMIZED style bit set, and the MS_VERTICALFLIP style bit of the system menu window is set.
- Window is maximized from a restored state.
 - Title-button windows:

The MAX button window is replaced with a RESTORE button window (the MIN button window is unaltered).
 - System menu window:

The RESTORE menu entry is enabled, the MAXIMIZE menu entry is disabled.
 - Other changes:

The frame window has the WS_MAXIMIZED style bit set.

WM_MINMAXFRAME (in Frame Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_MINMAXFRAME (in Frame Controls) - Related Messages

Related Messages

- [WM_MINMAXFRAME](#)
-

WM_MINMAXFRAME (in Frame Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

Frame Control Window Messages

This section describes the frame control window procedure actions on receiving the following messages.

WM_ACTIVATE (in Frame Controls)

WM_ACTIVATE (in Frame Controls) - Syntax

For the cause of this message, see [WM_ACTIVATE](#).

For a description of the parameters, see [WM_ACTIVATE](#).

WM_ACTIVATE (in Frame Controls) - Remarks

The frame control window procedure responds to this message by first sending a [TBM_SETHILITE](#) message to the FID_TITLEBAR control, if it exists, to highlight or unhighlight the title bar. If the style is FCF_DLGBORDER, the border is redrawn in either highlighted or unhighlighted state, as necessary.

It then sends the [WM_ACTIVATE](#) message to the FID_CLIENT window.

Then it sets *uiReserved* to 0.

WM_ACTIVATE (in Frame Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *uiReserved* to 0.

WM_ACTIVATE (in Frame Controls) - Related Messages

Related Messages

- [WM_ACTIVATE](#)

WM_ACTIVATE (in Frame Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_ADJUSTFRAMEPOS

WM_ADJUSTFRAMEPOS Field - pswp

pswp ([PSWP](#))

New frame window state.

This points to a [SWP](#) structure.

The structure has been filled in by the [WinSetWindowPos](#) or [WinSetMultWindowPos](#) functions with the proposed move or size data for the frame window.

WM_ADJUSTFRAMEPOS Field - hsavewphsvwp

hsavewphsvwp ([HSAVEWP](#))
Identifier of the frame window repositioning process.

WM_ADJUSTFRAMEPOS Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_ADJUSTFRAMEPOS - Parameters

pswp ([PSWP](#))
New frame window state.

This points to a [SWP](#) structure.

The structure has been filled in by the [WinSetWindowPos](#) or [WinSetMultWindowPos](#) functions with the proposed move or size data for the frame window.

hsavewphsvwp ([HSAVEWP](#))
Identifier of the frame window repositioning process.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_ADJUSTFRAMEPOS - Syntax

This message is sent to a frame window whose position or size is to be adjusted.

```
param1  
    PSWP    pswp          /* New frame window state. */  
  
param2  
    HSAVEWP hsavewphsvwp /* Identifier of the frame window repositioning process. */
```

WM_ADJUSTFRAMEPOS - Remarks

When a [WinSetWindowPos](#) or [WinSetMultWindowPos](#) function involves adjusting the position or size of a frame window, a WM_ADJUSTFRAMEPOS message is sent to the frame window.

The frame control processes the message by informing all the windows in its owner hierarchy, that is all the windows owned by the frame and all the windows owned by them and so on, by sending each a [WM_OWNERPOSCHANGE](#) message. Each window receiving the a [WM_OWNERPOSCHANGE](#) message is expected to modify the [SWP](#) structure provided as the first parameter in the message to the appropriate values relative to the new position and/or size of its owner, whose new position and size is specified in a [SWP](#) structure provided as the second parameter in the message.

In this way the frame control can determine the state changes to be made to all the windows in its owner hierarchy, in accordance with the values specified in the [SWP](#) structure referenced by the *pswp* parameter. The rules for changing the state of these owned windows are:

SWP_SIZE and SWP_MOVE

The owned window is moved relative to the top left corner of its owner.

SWP_SHOW

The visibility state of an owned window is changed to agree with that of their owner.

SWP_MINIMIZE

An owned window is made invisible when the owner is minimized.

SWP_MAXIMIZE and SWP_RESTORE

An owned window that was previously made invisible when the owner was minimized is made visible.

The frame window coordinates the repositioning of the frame window and all its owned windows, by using the [WinSaveWindowPos](#) function to associate those windows whose states are to change with the identifier of the frame window repositioning process, that is the *hsavewphswnp* parameter. Eventually, the state changes to be made to the owned windows are contained in the array of [SWP](#) structures identified by the parameter.

If the frame window is subclassed, this message must then be passed to the superclass window procedure for processing. The superclass window procedure is the window procedure of the window before it was subclassed. This message is passed along the chain of window procedures and is eventually processed by the system frame window procedure.

WM_ADJUSTFRAMEPOS - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_ADJUSTFRAMEPOS - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

WM_BUTTON1DBLCLK (in Frame Controls)

WM_BUTTON1DBLCLK (in Frame Controls) - Syntax

For the cause of this message, see [WM_BUTTON1DBLCLK](#).

For a description of the parameters, see [WM_BUTTON1DBLCLK](#).

WM_BUTTON1DBLCLK (in Frame Controls) - Default Processing

If the frame is minimized, the frame control window procedure causes the frame window to return to its previous state. Otherwise, the message is handled like a [WM_BUTTON1DOWN](#) message.

WM_BUTTON1DBLCLK (in Frame Controls) - Related Messages

Related Messages

- [WM_BUTTON1DBLCLK](#)
-

WM_BUTTON1DBLCLK (in Frame Controls) - Topics

Select an item:

[Syntax](#)

[Default Processing](#)

[Related Messages](#)

[Glossary](#)

WM_BUTTON2DBLCLK (in Frame Controls)

WM_BUTTON2DBLCLK (in Frame Controls) - Syntax

For the cause of this message, see [WM_BUTTON2DBLCLK](#).

For a description of the parameters, see [WM_BUTTON2DBLCLK](#).

WM_BUTTON2DBLCLK (in Frame Controls) - Default Processing

The frame control window procedure processes this message identically to [WM_BUTTON1DBLCLK \(in Frame Controls\)](#).

WM_BUTTON2DBLCLK (in Frame Controls) - Related Messages

Related Messages

- [WM_BUTTON2DBLCLK](#)
-

WM_BUTTON2DBLCLK (in Frame Controls) - Topics

Select an item:

[Syntax](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_BUTTON1DOWN (in Frame Controls)

WM_BUTTON1DOWN (in Frame Controls) - Syntax

For the cause of this message, see [WM_BUTTON1DOWN](#).

For a description of the parameters, see [WM_BUTTON1DOWN](#).

WM_BUTTON1DOWN (in Frame Controls) - Remarks

This message is posted to the application queue associated with the window that is to receive the pointer button information.

WM_BUTTON1DOWN (in Frame Controls) - Default Processing

The frame control window procedure responds to this message by issuing the [WinSetActiveWindow](#) function and sets *rc* to TRUE. If this is over a part of the window that does not have a frame control, it issues a [WinSetActiveWindow](#) function. If the click is over the size border,

this window begins tracking by sending a [WM_TRACKFRAME](#) message to itself. If the click is not over the size border, this message is passed on.

WM_BUTTON1DOWN (in Frame Controls) - Related Messages

Related Messages

- [WM_BUTTON1DOWN](#)
-

WM_BUTTON1DOWN (in Frame Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_BUTTON2DOWN (in Frame Controls)

WM_BUTTON2DOWN (in Frame Controls) - Syntax

For the cause of this message, see [WM_BUTTON2DOWN](#).

For a description of the parameters, see [WM_BUTTON2DOWN](#).

WM_BUTTON2DOWN (in Frame Controls) - Remarks

This message is posted to the application queue associated with the window that is to receive the pointer button information.

WM_BUTTON2DOWN (in Frame Controls) - Default Processing

The frame control window procedure processes this message identically to [WM_BUTTON1DOWN \(in Frame Controls\)](#).

WM_BUTTON2DOWN (in Frame Controls) - Related Messages

Related Messages

- [WM_BUTTON2DOWN](#)

WM_BUTTON2DOWN (in Frame Controls) - Topics

Select an item:

[Syntax](#)

[Remarks](#)

[Default Processing](#)

[Related Messages](#)

[Glossary](#)

WM_BUTTON1UP (in Frame Controls)

WM_BUTTON1UP (in Frame Controls) - Syntax

For the cause of this message, see [WM_BUTTON1UP](#).

For a description of the parameters, see [WM_BUTTON1UP](#).

WM_BUTTON1UP (in Frame Controls) - Remarks

This message is posted to the application queue associated with the window that is to receive the pointer button information.

WM_BUTTON1UP (in Frame Controls) - Default Processing

The frame control window procedure responds to this message by issuing the [WinSetActiveWindow](#) function and sets *rc* to TRUE. If the window is not minimized, this message is not processed. If the frame is minimized, this message causes the system menu to pop up.

WM_BUTTON1UP (in Frame Controls) - Related Messages

Related Messages

- [WM_BUTTON1UP](#)

WM_BUTTON1UP (in Frame Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_BUTTON2UP (in Frame Controls)

WM_BUTTON2UP (in Frame Controls) - Syntax

For the cause of this message, see [WM_BUTTON2UP](#).

For a description of the parameters, see [WM_BUTTON2UP](#).

WM_BUTTON2UP (in Frame Controls) - Remarks

This message is posted to the application queue associated with the window that is to receive the pointer button information.

WM_BUTTON2UP (in Frame Controls) - Default Processing

The frame control window procedure processes this message identically to [WM_BUTTON1UP \(in Frame Controls\)](#).

WM_BUTTON2UP (in Frame Controls) - Related Messages

Related Messages

- [WM_BUTTON2UP](#)

WM_BUTTON2UP (in Frame Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_CALCFRAMERECT (in Frame Controls)

WM_CALCFRAMERECT (in Frame Controls) - Syntax

For the cause of this message, see [WM_CALCFRAMERECT](#).

For a description of the parameters, see [WM_CALCFRAMERECT](#).

WM_CALCFRAMERECT (in Frame Controls) - Remarks

Frame control calculates the appropriate rectangle, taking into account byte alignment, or nonbyte alignment if FCF_NOBYTEALIGN is specified.

WM_CALCFRAMERECT (in Frame Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_CALCFRAMERECT (in Frame Controls) - Related Messages

Related Messages

- [WM_CALCFRAMERECT](#)
-

WM_CALCFRAMERECT (in Frame Controls) - Topics

Select an item:

[Syntax](#)

[Remarks](#)

[Default Processing](#)

[Related Messages](#)

[Glossary](#)

WM_CHAR (in Frame Controls)

WM_CHAR (in Frame Controls) - Syntax

This message is sent by controls to their owner window if they do not process the key stroke themselves. It is the most common means by which the input focus is switched around the various controls in a dialog box.

For a description of the parameters, see [WM_CHAR](#).

WM_CHAR (in Frame Controls) - Default Processing

The frame control window procedure responds to this message as follows:

- If the message contains a valid VK_ value, that value is processed before any valid character in the message.
 - If the character matches a mnemonic in the text of a button or static control child window, the focus is set to that window.
 - If the character is Tab or Backtab, the focus is set to the next or previous tabstop window.
 - If the character is Up or Left Arrow, the focus is set to the previous item in the group.
 - If the character is Down or Right Arrow, the focus is set to the next item in the group.
 - If the Enter key is pressed, a [WM_COMMAND](#) message is posted to itself, containing the identity of the button with the focus, or, if none, the identity of the default push button.
 - If the Escape key is pressed, a [WM_COMMAND](#) message is posted to itself with the command value DID_CANCEL.
-

WM_CHAR (in Frame Controls) - Related Messages

Related Messages

- [WM_CHAR](#)

WM_CHAR (in Frame Controls) - Topics

Select an item:

[Syntax](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_CLOSE (in Frame Controls)

WM_CLOSE (in Frame Controls) - Syntax

For the cause of this message, see [WM_CLOSE](#).

For a description of the parameters, see [WM_CLOSE](#).

WM_CLOSE (in Frame Controls) - Remarks

Frame control sends this message to the client window (FID_CLIENT) if it exists, otherwise it calls the [WinDefWindowProc](#) function.

WM_CLOSE (in Frame Controls) - Default Processing

The default window procedure posts a [WM_QUIT](#) message to the appropriate queue and sets *ulReserved* to 0.

WM_CLOSE (in Frame Controls) - Related Messages

Related Messages

- [WM_CLOSE](#)
-

WM_CLOSE (in Frame Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_COMMAND

WM_COMMAND - Syntax

For the cause of this message, see [WM_COMMAND](#).

For a description of the parameters, see [WM_COMMAND](#).

WM_COMMAND - Default Processing

The Frame Control window procedure responds to this message by sending it the client window if it exists, otherwise the message is thrown away.

WM_COMMAND - Topics

Select an item:

[Syntax](#)
[Default Processing](#)
[Glossary](#)

WM_DRAWITEM (in Frame Controls)

WM_DRAWITEM (in Frame Controls) - Syntax

For the cause of this message, see [WM_DRAWITEM](#).

For a description of the parameters, see [WM_DRAWITEM](#).

WM_DRAWITEM (in Frame Controls) - Remarks

The identity of the top-level action-bar menu that generated this message is found. If the identity is FID_MENU, the message is passed to the window with identity FID_CLIENT.

WM_DRAWITEM (in Frame Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_DRAWITEM (in Frame Controls) - Related Messages

Related Messages

- [WM_DRAWITEM](#)
-

WM_DRAWITEM (in Frame Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_ERASEBACKGROUND

WM_ERASEBACKGROUND Field - hpsFrame

hpsFrame ([HPS](#))

Presentation-space handle for the frame window.

WM_ERASEBACKGROUND Field - pprcPaint

pprcPaint ([PRECTL](#))
Rectangle structure of rectangle to be painted.

This points to a [RECTL](#) structure.

WM_ERASEBACKGROUND Return Value - rc

rc ([BOOL](#))
Processed indicator.

TRUE	If a FID_CLIENT window exists, the area of the frame covered by the FID_CLIENT window is erased in the system-window background color. If no FID_CLIENT window exists, the entire frame window is erased in the system-window background color.
FALSE	The client window did process the message.

WM_ERASEBACKGROUND - Parameters

hpsFrame ([HPS](#))
Presentation-space handle for the frame window.

pprcPaint ([PRECTL](#))
Rectangle structure of rectangle to be painted.

This points to a [RECTL](#) structure.

rc ([BOOL](#))
Processed indicator.

TRUE	If a FID_CLIENT window exists, the area of the frame covered by the FID_CLIENT window is erased in the system-window background color. If no FID_CLIENT window exists, the entire frame window is erased in the system-window background color.
FALSE	The client window did process the message.

WM_ERASEBACKGROUND - Syntax

This message causes a client window to be filled with the background, should this be appropriate.

param1

```
    HPS    hpsFrame    /* Presentation-space handle for the frame window. */  
param2  
    PRECTL pprcPaint /* Rectangle structure of rectangle to be painted. */
```

WM_ERASEBACKGROUND - Remarks

The frame window procedure processes this message in the following manner:

1. The frame window sends this message to the client in response to the frame [WM_PAINT](#) message, with the presentation-space handle of the frame window (obtained from [WinBeginPaint](#)).
2. If the client window returns TRUE, the frame window procedure erases the rectangle of the frame window covered by the client window, by filling it with the system color SCLR_WINDOW.
3. If the client window returns FALSE, no action is taken. This is the default behavior, as [WinDefWindowProc](#) returns FALSE if passed this message.
4. Also, the client window can use the presentation-space handle passed in this message to selectively erase parts of the screen. If the client window processes the message in this way, FALSE should be returned to avoid the erasure being done automatically by the frame window procedure.

It should be noted again that the presentation space is *not* a client window presentation space; it is a presentation space for the frame window returned by [WinBeginPaint](#), that is, a cached presentation space in frame (not client) window coordinates, clipped to the area of the frame that needs to be updated (possibly including areas outside the client window).

WM_ERASEBACKGROUND - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_ERASEBACKGROUND - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_FLASHWINDOW

WM_FLASHWINDOW Field - usFlash

usFlash (**USHORT**)
Flash indicator.

TRUE	Start the window border flashing
FALSE	Stop the window border flashing.

WM_FLASHWINDOW Field - ulReserved

ulReserved (**ULONG**)
Reserved value, should be 0.

WM_FLASHWINDOW Return Value - rc

rc (**BOOL**)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WM_FLASHWINDOW - Parameters

usFlash (**USHORT**)
Flash indicator.

TRUE	Start the window border flashing
FALSE	Stop the window border flashing.

ulReserved (**ULONG**)
Reserved value, should be 0.

rc (**BOOL**)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WM_FLASHWINDOW - Syntax

An application has issued a [WinFlashWindow](#) function.

```
param1
    USHORT    usFlash        /* Flash indicator. */

param2
    ULONG     ulReserved    /* Reserved value, should be 0. */
```

WM_FLASHWINDOW - Default Processing

The frame control window procedure responds to this message from an application by starting or stopping the flashing of the window border, and by setting *rc* as appropriate.

WM_FLASHWINDOW - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Default Processing](#)

[Glossary](#)

WM_FOCUSCHANGE (in Frame Controls)

WM_FOCUSCHANGE (in Frame Controls) - Syntax

For the cause of this message, see [WM_FOCUSCHANGE](#).

For a description of the parameters, see [WM_FOCUSCHANGE](#).

WM_FOCUSCHANGE (in Frame Controls) - Remarks

The frame control responds to this message by sending the other messages depending on the value of the *fsFocusChange* parameter. These messages, if sent, are sent in the following order:

1. [WM_SETFOCUS](#) to the window losing the focus.
2. [WM_SETSELECTION](#) to the windows losing their selection.

3. [WM_ACTIVATE](#) to the windows being deactivated.
4. [WM_ACTIVATE](#) to the windows being activated.
5. [WM_SETSELECTION](#) to the windows being selected.
6. [WM_SETFOCUS](#) to the window receiving the focus.

WM_FOCUSCHANGE (in Frame Controls) - Default Processing

The default window procedure sends this message to either the owner, if one exists, or to the parent of the window, if it is not the desktop window, otherwise it sets *uReserved* to 0.

WM_FOCUSCHANGE (in Frame Controls) - Related Messages

Related Messages

- [WM_FOCUSCHANGE](#)

WM_FOCUSCHANGE (in Frame Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_FORMATFRAME (in Frame Controls)

WM_FORMATFRAME (in Frame Controls) - Syntax

For the cause of this message, see [WM_FORMATFRAME](#).

For a description of the parameters, see [WM_FORMATFRAME](#).

WM_FORMATFRAME (in Frame Controls) - Remarks

Applications that subclass frame controls may find that the frame is already subclassed; the number of frame controls is variable.

The [WM_FORMATFRAME](#) and [WM_QUERYFRAMECTLCOUNT](#) messages must always be subclassed by calling the previous window procedure and modifying its result.

WM_FORMATFRAME (in Frame Controls) - Default Processing

The [SWP](#) structure for the FID_CLIENT frame control, if present, is the last element of the *pswp* parameter, unless additional frame controls are added by subclassing; the [SWP](#) structures for these follow that for FID_CLIENT if present. The frame control window procedure first sends the message to the FID_CLIENT window. If FID_CLIENT returns *ccount* to indicate that the message has been processed, no additional processing is performed.

If not processed by the client, the frame control window procedure calculates the size and position of all the standard frame controls.

WM_FORMATFRAME (in Frame Controls) - Related Messages

Related Messages

- [WM_FORMATFRAME](#)

WM_FORMATFRAME (in Frame Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_INITMENU (in Frame Controls)

WM_INITMENU (in Frame Controls) - Syntax

For the cause of this message, see [WM_INITMENU](#).

For a description of the parameters, see [WM_INITMENU](#).

WM_INITMENU (in Frame Controls) - Remarks

The identity of the top-level action-bar menu that generated this message is found. If the identity is FID_MENU, the message is passed to the window with identity FID_CLIENT. If the identity is FID_SYSMENU the system menu state is initialized according to the current state of the window.

WM_INITMENU (in Frame Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_INITMENU (in Frame Controls) - Related Messages

Related Messages

- [WM_INITMENU](#)

WM_INITMENU (in Frame Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_MEASUREITEM (in Frame Controls)

WM_MEASUREITEM (in Frame Controls) - Syntax

For the cause of this message, see [WM_MEASUREITEM](#).

For a description of the parameters, see [WM_MEASUREITEM](#).

WM_MEASUREITEM (in Frame Controls) - Remarks

The identity of the top-level action bar menu that generated this message is found. If the identity is FID_MENU, the message is passed to the window with identity FID_CLIENT.

WM_MEASUREITEM (in Frame Controls) - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sHeight* to the default value of 0.

WM_MEASUREITEM (in Frame Controls) - Related Messages

Related Messages

- [WM_MEASUREITEM](#)

WM_MEASUREITEM (in Frame Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_MENUSELECT (in Frame Controls)

WM_MENUSELECT (in Frame Controls) - Syntax

For the cause of this message, see WM_MENUSELECT (in Frame Controls).

For a description of the parameters, see WM_MENUSELECT (in Frame Controls).

WM_MENUSELECT (in Frame Controls) - Remarks

The identity of the top-level action-bar menu that generated this message is found. If the identity is FID_MENU, the message is passed to the window with identity FID_CLIENT.

WM_MENUSELECT (in Frame Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to TRUE.

WM_MENUSELECT (in Frame Controls) - Related Messages

Related Messages

- [WM_MENUSELECT](#)

WM_MENUSELECT (in Frame Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_NEXTMENU (in Frame Controls)

WM_NEXTMENU (in Frame Controls) - Syntax

For the cause of this message, see [WM_NEXTMENU](#).

For a description of the parameters, see [WM_NEXTMENU](#).

WM_NEXTMENU (in Frame Controls) - Remarks

The frame control window procedure processes the message by returning the handle of the system menu window if *hwndMenu* is the handle of the main action bar window, or by returning the handle of the main action bar window if *hwndMenu* is the handle of the system menu window.

WM_NEXTMENU (in Frame Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *hwndNewMenu* to NULLHANDLE.

WM_NEXTMENU (in Frame Controls) - Related Messages

Related Messages

- [WM_NEXTMENU](#)
-

WM_NEXTMENU (in Frame Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_OWNERPOSCHANGE

WM_OWNERPOSCHANGE Field - ppswp

ppswp ([PSWP](#))

Owned window state.

This points to a [SWP](#) structure.

The receiver of this message is expected to alter this [SWP](#) parameter to the appropriate values relative to the new position and/or size of its owner, whose new position and size is specified in a [SWP](#) structure in the *ppswpOwner* parameter.

WM_OWNERPOSCHANGE Field - ppswpOwner

ppswpOwner ([PSWP](#))

Owner window state.

This points to a [SWP](#) structure.

This represents the new position and size of the owner window.

WM_OWNERPOSCHANGE Return Value - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_OWNERPOSCHANGE - Parameters

ppswp ([PSWP](#))

Owned window state.

This points to a [SWP](#) structure.

The receiver of this message is expected to alter this [SWP](#) parameter to the appropriate values relative to the new position and/or size of its owner, whose new position and size is specified in a [SWP](#) structure in the *ppswpOwner* parameter.

ppswpOwner ([PSWP](#))

Owner window state.

This points to a [SWP](#) structure.

This represents the new position and size of the owner window.

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_OWNERPOSCHANGE - Syntax

This message is sent by a frame window processing the [WM_ADJUSTFRAMEPOS](#) message.

```
param1
    PSWP    ppswp        /* Owned window state. */

param2
    PSWP    ppswpOwner    /* Owner window state. */
```

WM_OWNERPOSCHANGE - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_OWNERPOSCHANGE - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Default Processing](#)

[Glossary](#)

WM_PAINT (in Frame Controls)

WM_PAINT (in Frame Controls) - Syntax

For the cause of this message, see [WM_PAINT](#).

For a description of the parameters, see [WM_PAINT](#).

WM_PAINT (in Frame Controls) - Default Processing

The frame is redrawn as governed by the FCF_BORDER or FCF_DLGBORDER style. A [WM_ERASEBACKGROUND](#) message is sent to FID_CLIENT window, and if it returns FALSE, then the FID_CLIENT window is erased to the system-provided window background color and sets *ulReserved* to 0.

WM_PAINT (in Frame Controls) - Related Messages

Related Messages

- [WM_PAINT](#)
-

WM_PAINT (in Frame Controls) - Topics

Select an item:

[Syntax](#)

[Default Processing](#)

WM_QUERYBORDERSIZE

WM_QUERYBORDERSIZE Field - pSize

pSize ([PWPOINT](#))
Width and height of size border control.

This points to a [POINTL](#) structure, that is used to hold the width in the *x* parameter and the height in the *y* parameter.

WM_QUERYBORDERSIZE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_QUERYBORDERSIZE Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WM_QUERYBORDERSIZE - Parameters

pSize ([PWPOINT](#))
Width and height of size border control.

This points to a [POINTL](#) structure, that is used to hold the width in the *x* parameter and the height in the *y* parameter.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WM_QUERYBORDERSIZE - Syntax

This message is sent to the frame window to determine the width and height of the border of the window.

```
param1  
    PWPOINT pSize    /* Width and height of size border control. */  
  
param2  
    ULONG   ulReserved /* Reserved value, should be 0. */
```

WM_QUERYBORDERSIZE - Remarks

The frame window responds to this message by returning the width and height of its border in the *pSize* parameter, as follows:

- SV_CX/CYSIZEBORDER if FCF_SIZEBORDER is specified
 - SV_CX/CYDLGFRAME if FCF_DLGBORDER is specified
 - SV_CX/CYBORDER if FS_BORDER is specified.
-

WM_QUERYBORDERSIZE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

WM_QUERYBORDERSIZE - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

WM_QUERYCONVERTPOS (in Frame Controls)

WM_QUERYCONVERTPOS (in Frame Controls) - Syntax

For the cause of this message, see [WM_QUERYCONVERTPOS](#).

For a description of the parameters, see [WM_QUERYCONVERTPOS](#).

WM_QUERYCONVERTPOS (in Frame Controls) - Remarks

The frame control window procedure returns QCP_NOCONVERT.

WM_QUERYCONVERTPOS (in Frame Controls) - Default Processing

For the default window procedure processing of this message see [WM_QUERYCONVERTPOS](#)

WM_QUERYCONVERTPOS (in Frame Controls) - Related Messages

Related Messages

- [WM_QUERYCONVERTPOS](#)
-

WM_QUERYCONVERTPOS (in Frame Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_QUERYFOCUSCHAIN

WM_QUERYFOCUSCHAIN Field - fsCmd

fsCmd ([USHORT](#))

Command to be performed.

This field contains a flag to indicate what action is to be performed:

QFC_NEXTINCHAIN

Return the next window in the focus chain.

The *hwndParent* parameter is not used.

QFC_ACTIVE

Return the handle of the frame window that would be activated or deactivated, if this window gains or loses the focus.

The window handle returned is a child of the window specified by the *hwndParent* parameter.

QFC_FRAME

Return the handle of the first frame window associated with this window.

The *hwndParent* parameter is not used.

QFC_SELECTACTIVE

Return the handle of the window from the group of owned windows to which this window belongs which either currently has the focus or, if no window has the focus, previously had the focus.

Return NULL, if no window in the owner group has had the focus.

The *hwndParent* parameter is not used.

QFC_PARTOFCHAIN

Return TRUE if the handle of the window identified by the *hwndParent* parameter is in the focus chain, otherwise return FALSE.

Because this message is passed along the focus chain, this is equivalent to returning TRUE, if the handle of the window receiving this message is *hwndParent* or to returning FALSE, if it is not.

WM_QUERYFOCUSCHAIN Field - hwndParent

hwndParent ([HWND](#))

Parent window.

WM_QUERYFOCUSCHAIN Return Value - hwndResult

hwndResult ([HWND](#))

Handle of the window requested.

0	No window handle exists for this case of the <i>fsCmd</i> parameter This value is also to be interpreted as FALSE for the case when the <i>fsCmd</i> is set to QFC_PARTOFCHAIN.
Other	Handle of the window requested. This value is also to be interpreted as TRUE for the cases when the <i>fsCmd</i> is set to QFC_PARTOFCHAIN.

WM_QUERYFOCUSCHAIN - Parameters

fsCmd (USHORT)

Command to be performed.

This field contains a flag to indicate what action is to be performed:

QFC_NEXTINCHAIN

Return the next window in the focus chain.

The *hwndParent* parameter is not used.

QFC_ACTIVE

Return the handle of the frame window that would be activated or deactivated, if this window gains or loses the focus.

The window handle returned is a child of the window specified by the *hwndParent* parameter.

QFC_FRAME

Return the handle of the first frame window associated with this window.

The *hwndParent* parameter is not used.

QFC_SELECTACTIVE

Return the handle of the window from the group of owned windows to which this window belongs which either currently has the focus or, if no window has the focus, previously had the focus.

Return NULL, if no window in the owner group has had the focus.

The *hwndParent* parameter is not used.

QFC_PARTOFCHAIN

Return TRUE if the handle of the window identified by the *hwndParent* parameter is in the focus chain, otherwise return FALSE.

Because this message is passed along the focus chain, this is equivalent to returning TRUE, if the handle of the window receiving this message is *hwndParent* or to returning FALSE, if it is not.

hwndParent (HWND)

Parent window.

hwndResult (HWND)

Handle of the window requested.

0	No window handle exists for this case of the <i>fsCmd</i> parameter This value is also to be interpreted as FALSE for the case when the <i>fsCmd</i> is set to QFC_PARTOFCHAIN.
Other	Handle of the window requested. This value is also to be interpreted as TRUE for the cases when the <i>fsCmd</i> is set to QFC_PARTOFCHAIN.

WM_QUERYFOCUSCHAIN - Syntax

This message is used to request the handle of a window in the focus chain.

```
param1
    USHORT   fsCmd      /* Command to be performed. */

param2
    HWND     hwndParent /* Parent window. */
```

WM_QUERYFOCUSCHAIN - Remarks

The frame control window procedure responds to this message by returning the appropriate window handle, as described under the *fsCmd* field.

WM_QUERYFOCUSCHAIN - Default Processing

The default window procedure takes the same action as the frame control window procedure.

WM_QUERYFOCUSCHAIN - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

WM_QUERYFRAMECTLCOUNT

WM_QUERYFRAMECTLCOUNT Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_QUERYFRAMECTLCOUNT Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_QUERYFRAMECTLCOUNT Return Value - sControlCount

sControlCount (SHORT)
Count of frame controls.

WM_QUERYFRAMECTLCOUNT - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

sControlCount (SHORT)
Count of frame controls.

WM_QUERYFRAMECTLCOUNT - Syntax

This message is sent to the frame window in response to the receipt of a WM_SIZE or a WM_UPDATEFRAME (in Frame Controls) message.

```
param1
    ULONG ulReserved    /* Reserved value, should be 0. */

param2
    ULONG ulReserved    /* Reserved value, should be 0. */
```

WM_QUERYFRAMECTLCOUNT - Remarks

By sending this message to itself, any procedures that subclass the frame window become aware that the number of frame controls is being calculated and include any special frame controls of the subclass in the count.

This count is used to allocate the appropriate number of [SWP](#) structures that are passed in the [WM_FORMATFRAME \(in Frame Controls\)](#) message.

WM_QUERYFRAMECTLCOUNT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sControlCount* to the default value of 0.

WM_QUERYFRAMECTLCOUNT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_QUERYFRAMEINFO

WM_QUERYFRAMEINFO Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_QUERYFRAMEINFO Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_QUERYFRAMEINFO Return Value - flFlags

flFlags (ULONG)

Frame information flags.

FI_FRAME

Identifies a frame window.

FI_OWNERHIDE

The frame window is hidden when its owner is hidden.

FI_NOMOVEWITHOWNER

The frame window does not move with its owner.

FI_ACTIVATEOK

The frame window may be activated. This means, for example, that the frame window is not disabled.

WM_QUERYFRAMEINFO - Parameters

ulReserved (ULONG)

Reserved value, should be 0.

ulReserved (ULONG)

Reserved value, should be 0.

flFlags (ULONG)

Frame information flags.

FI_FRAME

Identifies a frame window.

FI_OWNERHIDE

The frame window is hidden when its owner is hidden.

FI_NOMOVEWITHOWNER

The frame window does not move with its owner.

FI_ACTIVATEOK

The frame window may be activated. This means, for example, that the frame window is not disabled.

WM_QUERYFRAMEINFO - Syntax

This message enables an application to query information about frame windows.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

WM_QUERYFRAMEINFO - Remarks

This message can be used to query whether or not a particular window is a frame window.

WM_QUERYFRAMEINFO - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_QUERYFRAMEINFO - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_QUERYICON

WM_QUERYICON Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_QUERYICON Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_QUERYICON Return Value - hptrIcon

hptrIcon ([HPOINTER](#))
Handle to the icon.

WM_QUERYICON - Parameters

- ulReserved** ([ULONG](#))
Reserved value, should be 0.
- ulReserved** ([ULONG](#))
Reserved value, should be 0.
- hptrIcon** ([HPOINTER](#))
Handle to the icon.

WM_QUERYICON - Syntax

This message is sent to a frame window to query its associated icon.

```
param1
    ULONG      ulReserved /* Reserved value, should be 0. */

param2
    ULONG      ulReserved /* Reserved value, should be 0. */
```

WM_QUERYICON - Default Processing

The icon for the frame is returned.

WM_QUERYICON - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_QUERYWINDOWPARAMS (in Frame Controls)

WM_QUERYWINDOWPARAMS (in Frame Controls) - Syntax

This message occurs when an application queries the frame control window parameters.

For a description of the parameters, see [WM_QUERYWINDOWPARAMS](#).

WM_QUERYWINDOWPARAMS (in Frame Controls) - Default Processing

The frame control window procedure queries the appropriate window parameters in accordance with *pwndparams* and sets *rc* to TRUE if the operation is successful, otherwise to FALSE.

The window text of a frame control is obtained by sending this message to its FID_TITLEBAR.

WM_QUERYWINDOWPARAMS (in Frame Controls) - Related Messages

Related Messages

- [WM_QUERYWINDOWPARAMS](#)
-

WM_QUERYWINDOWPARAMS (in Frame Controls) - Topics

Select an item:

[Syntax](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_SETBORDERSIZE

WM_SETBORDERSIZE Field - uscx

uscx ([USHORT](#))
Width of border.

WM_SETBORDERSIZE Field - uscy

uscy (**USHORT**)
Height of border.

WM_SETBORDERSIZE Return Value - rc

rc (**BOOL**)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WM_SETBORDERSIZE - Parameters

uscx (**USHORT**)
Width of border.

uscy (**USHORT**)
Height of border.

rc (**BOOL**)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WM_SETBORDERSIZE - Syntax

This message is sent to the frame window to change the width and height of the border.

```
param1  
    USHORT  uscx    /* Width of border. */  
  
param2  
    USHORT  uscy    /* Height of border. */
```

WM_SETBORDERSIZE - Remarks

The frame control sets the width and height to *lscx* and *lscy* respectively.

WM_SETBORDERSIZE - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_SETBORDERSIZE - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_SETICON

WM_SETICON Field - hptrIcon

hptrIcon ([HPOINTER](#))
New icon handle.

WM_SETICON Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_SETICON Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WM_SETICON - Parameters

hptrIcon (HPOINTER)
New icon handle.

ulReserved (ULONG)
Reserved value, should be 0.

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WM_SETICON - Syntax

This message is sent to a frame window to set its associated icon.

```
param1
    HPOINTER  hptrIcon    /* New icon handle. */

param2
    ULONG     ulReserved  /* Reserved value, should be 0. */
```

WM_SETICON - Default Processing

The icon for the frame is set.

WM_SETICON - Topics

Select an item:

WM_SETWINDOWPARAMS (in Frame Controls)

WM_SETWINDOWPARAMS (in Frame Controls) - Syntax

This message occurs when an application sets or changes the frame control window parameters.

For a description of the parameters, see [WM_SETWINDOWPARAMS](#).

WM_SETWINDOWPARAMS (in Frame Controls) - Default Processing

The frame control window procedure sets the appropriate window parameters in accordance with *pwndparams* and sets *rc* to TRUE if the operation is successful, otherwise to FALSE.

The window text of a frame control is set by sending this message to its FID_TITLEBAR.

WM_SETWINDOWPARAMS (in Frame Controls) - Related Messages

Related Messages

- [WM_SETWINDOWPARAMS](#)
-

WM_SETWINDOWPARAMS (in Frame Controls) - Topics

Select an item:

WM_SIZE (in Frame Controls)

WM_SIZE (in Frame Controls) - Syntax

For the cause of this message, see [WM_SIZE](#).
For a description of the parameters, see [WM_SIZE](#).

WM_SIZE (in Frame Controls) - Default Processing

The frame control window procedure responds to this message by sending a [WM_FORMATFRAME \(in Frame Controls\)](#) message to itself and by setting *ulReserved* to 0.

WM_SIZE (in Frame Controls) - Related Messages

Related Messages

- [WM_SIZE](#)
-

WM_SIZE (in Frame Controls) - Topics

Select an item:
[Syntax](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_SYSCOMMAND

WM_SYSCOMMAND Field - uscmd

uscmd (USHORT)

Command value.

The frame control takes the action described on these *uscmd* values:

SC_SIZE	Sends a WM_TRACKFRAME (in Frame Controls) to the frame window.
SC_MOVE	Sends a WM_TRACKFRAME (in Frame Controls) to the frame window.
SC_MINIMIZE	If a control with the identifier FID_MINMAX is present, minimizes the frame window; otherwise nothing is done.
SC_MAXIMIZE	If a control with the identifier FID_MINMAX is present, maximizes the frame window; otherwise nothing is done. When a window is moved or sized in the normal way at least one border should remain on the screen. When a window is maximized and the maximum size is as large as the screen, all borders should be positioned just outside the screen.
SC_RESTORE	If a control with the identifier FID_MINMAX is present, restores a maximized or minimized frame window to its previous size and position; otherwise nothing is done.
SC_NEXT	Cycles the active window status to the next main window.
SC_APPMENU	Sends a MM_STARTMENU message to the control with the identifier FID_MENU.
SC_SYSMENU	Sends a MM_STARTMENU message to the control with the identifier FID_SYSMENU.
SC_CLOSE	If Close is not enabled in the system menu, this message is ignored. Otherwise the frame posts a WM_CLOSE message to the client if it exists or to itself, if not.
SC_NEXTFRAME	The next frame window that is a child of the desktop window is activated.
SC_NEXTWINDOW	The next window with the same owner window is activated.
SC_TASKMANAGER	The Task List is activated.
SC_HELPEXTENDED	The frame manager sends HM_EXT_HELP to the associated Help Manager Object Window. If there is no such associated window, the original message is sent to the client.
SC_HELPKEYS	The frame manager sends HM_KEYS_HELP to the associated Help Manager Object Window. If there is no such associated window, the original message is sent to the client.
SC_HELPINDEX	The frame manager sends HM_HELP_INDEX to the associated Help Manager Object Window. If there is no such associated window, the original message is sent to the client.
SC_HIDE	Sets the visibility state of the frame window to off causing it to appear hidden or invisible.

WM_SYSCOMMAND Field - **ussource**

ussource (USHORT)

Source type.

Identifies the type of control:

CMDSRC_PUSHBUTTON

Posted by a push-button control: *uscmd* is the window identifier of the push button.

CMDSRC_MENU

Posted by a menu control: *uscmd* is the identifier of the menu item.

CMDSRC_ACCELERATOR

Posted as the result of an accelerator: *uscmd* is the accelerator command value.

CMDSRC_OTHER

Other source: *uscmd* gives further control-specific information defined for each control type.

WM_SYSCOMMAND Field - fpointer

fpointer ([BOOL](#))

Pointing-device indicator.

TRUE

The message is posted as a result of a pointing-device operation.

FALSE

The message is posted as a result of a keyboard operation.

WM_SYSCOMMAND Parameter - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_SYSCOMMAND - Parameters

uscmd ([USHORT](#))

Command value.

The frame control takes the action described on these *uscmd* values:

SC_SIZE

Sends a [WM_TRACKFRAME](#) (in [Frame Controls](#)) to the frame window.

SC_MOVE

Sends a [WM_TRACKFRAME](#) (in [Frame Controls](#)) to the frame window.

SC_MINIMIZE

If a control with the identifier FID_MINMAX is present, minimizes the frame window; otherwise nothing is done.

SC_MAXIMIZE

If a control with the identifier FID_MINMAX is present, maximizes the frame window; otherwise nothing is done.

When a window is moved or sized in the normal way at least one border should remain on the screen. When a

window is maximized and the maximum size is as large as the screen, all borders should be positioned just outside the screen.

SC_RESTORE

If a control with the identifier FID_MINMAX is present, restores a maximized or minimized frame window to its previous size and position; otherwise nothing is done.

SC_NEXT

Cycles the active window status to the next main window.

SC_APPMENU

Sends a [MM_STARTMENU](#) message to the control with the identifier FID_MENU.

SC_SYSMENU

Sends a [MM_STARTMENU](#) message to the control with the identifier FID_SYSMENU.

SC_CLOSE

If Close is not enabled in the system menu, this message is ignored. Otherwise the frame posts a [WM_CLOSE](#) message to the client if it exists or to itself, if not.

SC_NEXTFRAME

The next frame window that is a child of the desktop window is activated.

SC_NEXTWINDOW

The next window with the same owner window is activated.

SC_TASKMANAGER

The Task List is activated.

SC_HELPEXTENDED

The frame manager sends HM_EXT_HELP to the associated Help Manager Object Window. If there is no such associated window, the original message is sent to the client.

SC_HELPKEYS

The frame manager sends HM_KEYS_HELP to the associated Help Manager Object Window. If there is no such associated window, the original message is sent to the client.

SC_HELPINDEX

The frame manager sends HM_HELP_INDEX to the associated Help Manager Object Window. If there is no such associated window, the original message is sent to the client.

SC_HIDE

Sets the visibility state of the frame window to off causing it to appear hidden or invisible.

ussource ([USHORT](#))

Source type.

Identifies the type of control:

CMDSRC_PUSHBUTTON

Posted by a push-button control: *uscmd* is the window identifier of the push button.

CMDSRC_MENU

Posted by a menu control: *uscmd* is the identifier of the menu item.

CMDSRC_ACCELERATOR

Posted as the result of an accelerator: *uscmd* is the accelerator command value.

CMDSRC_OTHER

Other source: *uscmd* gives further control-specific information defined for each control type.

fpointer ([BOOL](#))

Pointing-device indicator.

TRUE

The message is posted as a result of a pointing-device operation.

FALSE

The message is posted as a result of a keyboard operation.

uiReserved ([ULONG](#))

Reserved value, should be 0.

WM_SYSCOMMAND - Syntax

This message occurs when a control window has a significant event to notify to its owner, or when a key stroke has been translated by an accelerator table into a [WM_SYSCOMMAND](#).

```
param1
    USHORT    uscmd        /* Command value. */

param2
    USHORT    ussource     /* Source type. */
    BOOL      fpointer     /* Pointing-device indicator. */
```

WM_SYSCOMMAND - Remarks

This message is posted to the window procedure of the owner of the frame control. *uiReserved* is set to 0.

WM_SYSCOMMAND - Default Processing

The default window procedure takes no action on this message, other than to set *uiReserved* to 0.

WM_SYSCOMMAND - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_TRACKFRAME (in Frame Controls)

WM_TRACKFRAME (in Frame Controls) Field - fsTrackFlags

fsTrackFlags ([USHORT](#))
Tracking flags.

Contains a combination of one or more TF_* flags; for details, see the [TRACKINFO](#) data structure.

WM_TRACKFRAME (in Frame Controls) Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_TRACKFRAME (in Frame Controls) Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred, or the operation is terminated.

WM_TRACKFRAME (in Frame Controls) - Parameters

fsTrackFlags ([USHORT](#))
Tracking flags.

Contains a combination of one or more TF_* flags; for details, see the [TRACKINFO](#) data structure.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred, or the operation is terminated.

WM_TRACKFRAME (in Frame Controls) - Syntax

This message is sent to a frame window whenever it is to be moved or sized.

```
param1
    USHORT    fsTrackFlags    /* Tracking flags. */

param2
    ULONG     ulReserved      /* Reserved value, should be 0. */
```

WM_TRACKFRAME (in Frame Controls) - Remarks

The frame control window procedure responds to this message by causing a tracking rectangle to be drawn to move or size the window. For information, see the [WinTrackRect](#) function.

WM_TRACKFRAME (in Frame Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to TRUE.

WM_TRACKFRAME (in Frame Controls) - Related Messages

Related Messages

- [WM_TRACKFRAME](#)

WM_TRACKFRAME (in Frame Controls) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_TRANSLATEACCEL (in Frame Controls)

WM_TRANSLATEACCEL (in Frame Controls) - Syntax

For the cause of this message, see [WM_TRANSLATEACCEL](#).

For a description of the parameters, see [WM_TRANSLATEACCEL](#).

WM_TRANSLATEACCEL (in Frame Controls) - Remarks

The frame control window procedure processes the message by checking whether the character is in the accelerator table, by using the [WinTranslateAccel](#) function.

WM_TRANSLATEACCEL (in Frame Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_TRANSLATEACCEL (in Frame Controls) - Related Messages

Related Messages

- [WM_TRANSLATEACCEL](#)
-

WM_TRANSLATEACCEL (in Frame Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_TRANSLATEMNEMONIC (in Frame Controls)

WM_TRANSLATEMNEMONIC (in Frame Controls) - Syntax

For the cause of this message, see [WM_TRANSLATEMNEMONIC](#).

For a description of the parameters, see [WM_TRANSLATEMNEMONIC](#).

WM_TRANSLATEMNEMONIC (in Frame Controls) - Remarks

The frame control window procedure processes the message by sending it to the application menu window, that is, the window with the identity FID_MENU.

WM_TRANSLATEMNEMONIC (in Frame Controls) - Default Processing

For the default window procedure processing of this message, see [WM_TRANSLATEMNEMONIC](#).

WM_TRANSLATEMNEMONIC (in Frame Controls) - Related Messages

Related Messages

- [WM_TRANSLATEMNEMONIC](#)
-

WM_TRANSLATEMNEMONIC (in Frame Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_UPDATEFRAME (in Frame Controls)

WM_UPDATEFRAME (in Frame Controls) - Syntax

For the cause of this message, see [WM_UPDATEFRAME](#).

For a description of the parameters, see [WM_UPDATEFRAME](#).

WM_UPDATEFRAME (in Frame Controls) - Remarks

This message must be sent to the frame window whenever an application adds or removes one of the frame controls identified by the FCF_* flags. It must also be sent if the application adds or removes a submenu of the menu bar of the frame window.

The frame control window procedure first sends the message on to the FID_CLIENT window. The FID_CLIENT window might either reformat the frame window and set *rc* to TRUE, in which case the frame control window procedure takes no further action, or it might set *rc* to FALSE, in which case the frame control window procedure performs the reformatting.

If *//CreateFlags* contains FCF_SIZEBORDER, reformatting the frame window includes invalidating the area occupied by the size border.

The frame control window procedure sets *rc* to TRUE.

WM_UPDATEFRAME (in Frame Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to TRUE.

WM_UPDATEFRAME (in Frame Controls) - Related Messages

Related Messages

- [WM_UPDATEFRAME](#)
-

WM_UPDATEFRAME (in Frame Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

Help Manager Message Processing

This section describes the processing of messages sent by the Help Manager or applications in response to requests for help by the user.

Help Manager messages

The following messages are sent by the Help Manager to the application, or by the application to the Help Manager.

HM_ACTIONBAR_COMMAND

HM_ACTIONBAR_COMMAND Field - idCommand

idCommand (USHORT)

Identity of the action bar item that was selected.

HM_ACTIONBAR_COMMAND Field - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

HM_ACTIONBAR_COMMAND Return Value - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

HM_ACTIONBAR_COMMAND - Parameters

idCommand (USHORT)

Identity of the action bar item that was selected.

ulReserved (ULONG)

Reserved value, should be 0.

ulReserved (ULONG)

Reserved value, should be 0.

HM_ACTIONBAR_COMMAND - Syntax

This message is sent to the current active application window by the Help Manager to notify the application when the user selects a tailored action bar item.

```
param1
    USHORT  idCommand  /* Identity of the action bar item that was selected. */

param2
    ULONG   ulReserved /* Reserved value, should be 0. */
```

HM_ACTIONBAR_COMMAND - Default Processing

None.

HM_ACTIONBAR_COMMAND - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Default Processing](#)
 - [Glossary](#)

HM_CONTROL

HM_CONTROL Field - usReserve

usReserve ([USHORT](#))
Reserved value.

HM_CONTROL Field - controlres

controlres ([USHORT](#))
Res number of the control that was selected.

For author-defined push buttons, this is the res identification number that was specified with the push button tag (:pbutton.). For default push buttons, this is the res identification number defined in the PMHELP.H file.

HM_CONTROL Field - ulReserved

ulReserved (ULONG)
Reserved value.

HM_CONTROL Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_CONTROL - Parameters

usReserve (USHORT)
Reserved value.

controlres (USHORT)
Res number of the control that was selected.

For author-defined push buttons, this is the res identification number that was specified with the push button tag (:pbutton.). For default push buttons, this is the res identification number defined in the PMHELP.H file.

ulReserved (ULONG)
Reserved value.

ulReserved (ULONG)
Reserved value, should be 0.

HM_CONTROL - Syntax

This message is sent by the Help Manager to the child of the coverage window to add a control in the control area of a window.

```
param1
    USHORT  usReserve    /* Reserved value. */
    USHORT  controlres   /* Res number of the control that was selected. */

param2
    ULONG   ulReserved   /* Reserved value. */
```

HM_CONTROL - Remarks

If an application wants to filter any of the controls, it can subclass the child of the coverage window and intercept this message. If the application does not intercept this message, the Help Manager adds the control to the control area.

HM_CONTROL - Default Processing

None.

HM_CONTROL - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

HM_CREATE_HELP_TABLE

HM_CREATE_HELP_TABLE Field - pHELPTABLE

pHELPTABLE ([PHELPTABLE](#))

Help table.

This points to a help table structure; see [HELPTABLE](#).

HM_CREATE_HELP_TABLE Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

HM_CREATE_HELP_TABLE Return Value - rc

rc (ULONG)	Return code.
0	The procedure was successfully completed
Other	See the values of the <i>ulErrorCode</i> parameter of the HM_ERROR message.

HM_CREATE_HELP_TABLE - Parameters

pHELPTABLE (PHELPTABLE)	Help table.
	This points to a help table structure; see HELPTABLE .
ulReserved (ULONG)	Reserved value, should be 0.
rc (ULONG)	Return code.
0	The procedure was successfully completed
Other	See the values of the <i>ulErrorCode</i> parameter of the HM_ERROR message.

HM_CREATE_HELP_TABLE - Syntax

This message is sent by the application to give the Help Manager a new help table.

```
param1
    PHELPTABLE  pHELPTABLE  /*  Help table.  */

param2
    ULONG        ulReserved  /*  Reserved value, should be 0.  */
```

HM_CREATE_HELP_TABLE - Default Processing

None.

HM_CREATE_HELP_TABLE - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Glossary](#)

HM_DISMISS_WINDOW

HM_DISMISS_WINDOW Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

HM_DISMISS_WINDOW Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

HM_DISMISS_WINDOW Return Value - rc

rc ([ULONG](#))
Return code.

0	The help window was successfully removed
Other	There was no associated help window.

See also the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_DISMISS_WINDOW - Parameters

ulReserved ([ULONG](#))

Reserved value, should be 0.

ulReserved ([ULONG](#))

Reserved value, should be 0.

rc ([ULONG](#))

Return code.

0

The help window was successfully removed

Other

There was no associated help window.

See also the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_DISMISS_WINDOW - Syntax

This message tells the Help Manager to remove the active help window.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */
```

```
param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

HM_DISMISS_WINDOW - Remarks

If the user requests help from a primary or secondary window, and then interacts with the primary or secondary window without leaving help, the currently displayed help window might not be appropriate for the application window. This message gives the application the ability to remove that help window.

HM_DISMISS_WINDOW - Default Processing

None.

HM_DISMISS_WINDOW - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

HM_DISPLAY_HELP

HM_DISPLAY_HELP Field - idHelpPanelId

idHelpPanelId ([USHORT](#))
Identity of the help window.

This points to a [USHORT](#) data type.

For a value of the *usTypeFlag* parameter of HM_PANELNAME.

HM_DISPLAY_HELP Field - pszHelpPanelName

pszHelpPanelName ([PSZ](#))
Name of the help window.

This points to a string containing the name of the help window.

HM_DISPLAY_HELP Field - usTypeFlag

usTypeFlag ([USHORT](#))
Flag indicating how to interpret the first parameter.

HM_RESOURCEID
Indicates the *param1* points to the identity of the help window.

HM_PANELNAME
Indicates the *param1* points to the name of the help window.

HM_DISPLAY_HELP Return Value - rc

rc ([ULONG](#))
Return code.

0	The window was successfully displayed
Other	See the values of the <i>ulErrorCode</i> parameter of the HM_ERROR message.

HM_DISPLAY_HELP - Parameters

idHelpPanelId ([USHORT](#))

Identity of the help window.

This points to a [USHORT](#) data type.

For a value of the *usTypeFlag* parameter of HM_PANELNAME.

pszHelpPanelName ([PSZ](#))

Name of the help window.

This points to a string containing the name of the help window.

usTypeFlag ([USHORT](#))

Flag indicating how to interpret the first parameter.

HM_RESOURCEID

Indicates the *param1* points to the identity of the help window.

HM_PANELNAME

Indicates the *param1* points to the name of the help window.

rc ([ULONG](#))

Return code.

0	The window was successfully displayed
Other	See the values of the <i>ulErrorCode</i> parameter of the HM_ERROR message.

HM_DISPLAY_HELP - Syntax

This message tells the Help Manager to display a specific help window.

```
param1
    USHORT    idHelpPanelId    /* Identity of the help window. */
    PSZ       pszHelpPanelName /* Name of the help window. */

param2
    USHORT    usTypeFlag      /* Flag indicating how to interpret the first parameter. */
```

HM_DISPLAY_HELP - Remarks

param1 depends on the value of the *usTypeFlag* parameter.

HM_DISPLAY_HELP - Default Processing

None.

HM_DISPLAY_HELP - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

HM_ERROR

HM_ERROR Field - ulErrorCode

ulErrorCode ([ULONG](#))

Error code.

A constant describing the type of error that occurred. The application can also receive some of these error constants in the *ulReserved* parameter of messages it has sent to the Help Manager.

The error constants are:

HMERR_LOAD_DLL

The resource DLL was unable to be loaded.

HMERR_NO_FRAME_WND_IN_CHAIN

There is no frame window in the window chain from which to find or set the associated help instance.

HMERR_INVALID_ASSOC_APP_WND

The application window handle specified on the [WinAssociateHelpInstance](#) function is not a valid window handle.

HMERR_INVALID_ASSOC_HELP_INST

The help instance handle specified on the [WinAssociateHelpInstance](#) function is not a valid window handle.

HMERR_INVALID_DESTROY_HELP_INST

The window handle specified as the help instance to destroy is not of the help instance class.

HMERR_NO_HELP_INST_IN_CHAIN

The parent or owner chain of the application window specified does not have an associated help instance.

HMERR_INVALID_HELP_INSTANCE_HDL

The handle specified to be a help instance does not have the class name of a Help Manager instance.

HMERR_INVALID_QUERY_APP_WND
The application window specified on a [WinQueryHelpInstance](#) function is not a valid window handle.

HMERR_HELP_INST_CALLED_INVALID
The handle of the instance specified on a call to the Help Manager does not have the class name of a Help Manager instance.

HMERR_HELPTABLE_UNDEFINE
The application did not provide a help table for context-sensitive help.

HMERR_HELP_INSTANCE_UNDEFINE
The help instance handle specified is invalid.

HMERR_HELPITEM_NOT_FOUND
Context-sensitive help was requested but the ID of the main help item specified was not found in the help table.

HMERR_INVALID_HELPSUBITEM_SIZE
The help subtable item size is less than 2.

HMERR_HELPSUBITEM_NOT_FOUND
Context-sensitive help was requested but the ID of the help item specified was not found in the help subtable.

HMERR_INDEX_NOT_FOUND
The index is not in the library file.

HMERR_CONTENT_NOT_FOUND
The library file does not have any content.

HMERR_OPEN_LIB_FILE
The library file cannot be opened.

HMERR_READ_LIB_FILE
The library file cannot be read.

HMERR_CLOSE_LIB_FILE
The library file cannot be closed.

HMERR_INVALID_LIB_FILE
Improper library file provided.

HMERR_NO_MEMORY
Unable to allocate the requested amount of memory.

HMERR_ALLOCATE_SEGMENT
Unable to allocate a segment of memory for memory allocation requests from the Help Manager.

HMERR_FREE_MEMORY
Unable to free allocated memory.

HMERR_PANEL_NOT_FOUND
Unable to find the requested help window.

HMERR_DATABASE_NOT_OPEN
Unable to read the unopened database.

HM_ERROR Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

HM_ERROR Return Value - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

HM_ERROR - Parameters

ulErrorCode (ULONG)

Error code.

A constant describing the type of error that occurred. The application can also receive some of these error constants in the *ulReserved* parameter of messages it has sent to the Help Manager.

The error constants are:

HMERR_LOAD_DLL

The resource DLL was unable to be loaded.

HMERR_NO_FRAME_WND_IN_CHAIN

There is no frame window in the window chain from which to find or set the associated help instance.

HMERR_INVALID_ASSOC_APP_WND

The application window handle specified on the [WinAssociateHelpInstance](#) function is not a valid window handle.

HMERR_INVALID_ASSOC_HELP_INST

The help instance handle specified on the [WinAssociateHelpInstance](#) function is not a valid window handle.

HMERR_INVALID_DESTROY_HELP_INST

The window handle specified as the help instance to destroy is not of the help instance class.

HMERR_NO_HELP_INST_IN_CHAIN

The parent or owner chain of the application window specified does not have an associated help instance.

HMERR_INVALID_HELP_INSTANCE_HDL

The handle specified to be a help instance does not have the class name of a Help Manager instance.

HMERR_INVALID_QUERY_APP_WND

The application window specified on a [WinQueryHelpInstance](#) function is not a valid window handle.

HMERR_HELP_INST_CALLED_INVALID

The handle of the instance specified on a call to the Help Manager does not have the class name of a Help Manager instance.

HMERR_HELPTABLE_UNDEFINE

The application did not provide a help table for context-sensitive help.

HMERR_HELP_INSTANCE_UNDEFINE

The help instance handle specified is invalid.

HMERR_HELPITEM_NOT_FOUND

Context-sensitive help was requested but the ID of the main help item specified was not found in the help table.

HMERR_INVALID_HELPSUBITEM_SIZE

The help subtable item size is less than 2.

HMERR_HELPSUBITEM_NOT_FOUND

Context-sensitive help was requested but the ID of the help item specified was not found in the help subtable.

HMERR_INDEX_NOT_FOUND

The index is not in the library file.

HMERR_CONTENT_NOT_FOUND

The library file does not have any content.

HMERR_OPEN_LIB_FILE
The library file cannot be opened.

HMERR_READ_LIB_FILE
The library file cannot be read.

HMERR_CLOSE_LIB_FILE
The library file cannot be closed.

HMERR_INVALID_LIB_FILE
Improper library file provided.

HMERR_NO_MEMORY
Unable to allocate the requested amount of memory.

HMERR_ALLOCATE_SEGMENT
Unable to allocate a segment of memory for memory allocation requests from the Help Manager.

HMERR_FREE_MEMORY
Unable to free allocated memory.

HMERR_PANEL_NOT_FOUND
Unable to find the requested help window.

HMERR_DATABASE_NOT_OPEN
Unable to read the unopened database.

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

HM_ERROR - Syntax

This message notifies the application of an error caused by a user interaction.

```
param1
    ULONG   ulErrorCode /* Error code. */

param2
    ULONG   ulReserved /* Reserved value, should be 0. */
```

HM_ERROR - Remarks

There is no other way to communicate the error to the application since the user initiated communication, not the application. Other errors which occur when the application sends a message to the Help Manager are returned as the *ulReserved* parameter of the message.

The Help Manager does not display any error messages to the user. Instead, the Help Manager sends or returns all error notifications to the application so that it can display its own messages. This procedure ensures a consistent message interface for all user messages.

HM_ERROR - Default Processing

None.

HM_ERROR - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

HM_EXT_HELP

HM_EXT_HELP Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

HM_EXT_HELP Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

HM_EXT_HELP Return Value - rc

- rc** ([ULONG](#))
Return code.
- | | |
|-------|---|
| 0 | The extended help window was successfully displayed |
| Other | See the values of the <i>ulErrorCode</i> parameter of the HM_ERROR message. |

HM_EXT_HELP - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

rc (ULONG)
Return code.

- 0
The extended help window was successfully displayed
- Other
See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_EXT_HELP - Syntax

When the Help Manager receives this message, it displays the extended help window for the active application panel.

```
param1
    ULONG  ulReserved /* Reserved value, should be 0. */

param2
    ULONG  ulReserved /* Reserved value, should be 0. */
```

HM_EXT_HELP - Default Processing

None.

HM_EXT_HELP - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Default Processing](#)
 - [Glossary](#)

HM_EXT_HELP_UNDEFINED

HM_EXT_HELP_UNDEFINED Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_EXT_HELP_UNDEFINED Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_EXT_HELP_UNDEFINED Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_EXT_HELP_UNDEFINED - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

HM_EXT_HELP_UNDEFINED - Syntax

This message is sent to the application by the Help Manager to notify it that an extended help window has not been defined.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */
```

```
param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

HM_EXT_HELP_UNDEFINED - Remarks

When the extended help window is requested, the Help Manager searches the help table for its identity. If the extended help window identity associated with the current active window is zero, the Help Manager sends this message to the application to notify it that an extended help window has not been defined. The application then can:

- Ignore the request for help and not display a help window.
- Display its own window.
- Use the [HM_DISPLAY_HELP](#) message to tell the Help Manager to display a particular window.

HM_EXT_HELP_UNDEFINED - Default Processing

None.

HM_EXT_HELP_UNDEFINED - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

HM_GENERAL_HELP

HM_GENERAL_HELP Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

HM_GENERAL_HELP Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_GENERAL_HELP Return Value - rc

rc (ULONG)
Return code.

0	The general help window was successfully displayed.
Other	See the values of the <i>ulErrorCode</i> parameter of the HM_ERROR message.

HM_GENERAL_HELP - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

rc (ULONG)
Return code.

0	The general help window was successfully displayed.
Other	See the values of the <i>ulErrorCode</i> parameter of the HM_ERROR message.

HM_GENERAL_HELP - Syntax

When the Help Manager receives this message, it displays the general help window for the active application window.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

HM_GENERAL_HELP - Default Processing

None.

HM_GENERAL_HELP - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Glossary](#)

HM_GENERAL_HELP_UNDEFINED

HM_GENERAL_HELP_UNDEFINED Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

HM_GENERAL_HELP_UNDEFINED Field - ulReserved

ulReserved ([ULONG](#))
Reserved.

0
Reserved value, 0.

HM_GENERAL_HELP_UNDEFINED Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

HM_GENERAL_HELP_UNDEFINED - Parameters

ulReserved ([ULONG](#))

Reserved value, should be 0.

ulReserved ([ULONG](#))

Reserved.

0

Reserved value, 0.

ulReserved ([ULONG](#))

Reserved value, should be 0.

HM_GENERAL_HELP_UNDEFINED - Syntax

This message is sent to the application by the Help Manager to notify it that a general help window has not been defined.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */

param2
    ULONG ulReserved /* Reserved. */
```

HM_GENERAL_HELP_UNDEFINED - Remarks

When the general help window is requested, the Help Manager searches the help table for its identity. If the general help window identity associated with the current active window is zero, the Help Manager sends this message to the application to notify it that a general help window has not been defined. The application can then:

- Ignore the request for help and not display a help window.
- Display its own window.
- Use the [HM_DISPLAY_HELP](#) message to tell the Help Manager to display a particular window.

HM_GENERAL_HELP_UNDEFINED - Default Processing

None.

HM_GENERAL_HELP_UNDEFINED - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

HM_HELP_CONTENTS

HM_HELP_CONTENTS Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

HM_HELP_CONTENTS Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

HM_HELP_CONTENTS Return Value - rc

rc ([ULONG](#))
Return code.

0	The help contents window was successfully displayed.
Other	See the values of the <i>ulErrorCode</i> parameter of the HM_ERROR message.

HM_HELP_CONTENTS - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([ULONG](#))
Return code.

0
The help contents window was successfully displayed.

Other
See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_HELP_CONTENTS - Syntax

When the Help Manager receives this message, it displays the help contents window.

```
param1
    ULONG  ulReserved  /* Reserved value, should be 0. */

param2
    ULONG  ulReserved  /* Reserved value, should be 0. */
```

HM_HELP_CONTENTS - Default Processing

None.

HM_HELP_CONTENTS - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Default Processing](#)
 - [Glossary](#)
-

HM_HELP_INDEX

HM_HELP_INDEX Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_HELP_INDEX Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_HELP_INDEX Return Value - rc

rc (ULONG)
Return code.

0	The help index window was successfully displayed.
Other	See the values of the <i>ulErrorCode</i> parameter of the HM_ERROR message.

HM_HELP_INDEX - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

rc (ULONG)
Return code.

0	The help index window was successfully displayed.
Other	See the values of the <i>ulErrorCode</i> parameter of the HM_ERROR message.

HM_HELP_INDEX - Syntax

When the Help Manager receives this message, it displays the help index window.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */
param2
```



```
ULONG ulReserved /* Reserved value, should be 0. */
```

HM_HELP_INDEX - Default Processing

None.

HM_HELP_INDEX - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Default Processing](#)
 - [Glossary](#)

HM_HELPSUBITEM_NOT_FOUND

HM_HELPSUBITEM_NOT_FOUND Field - usContext

usContext ([USHORT](#))
Type of window on which help was requested.

HLPM_WINDOW	An application window
HLPM_FRAME	A frame window
HLPM_MENU	A menu window.

HM_HELPSUBITEM_NOT_FOUND Field - sTopic

sTopic ([SHORT](#))
Topic identifier.

For a value of the *usContext* parameter of HLPM_WINDOW or HLPM_FRAME:

window	Identity of the window containing the field on which help was requested.
menu	Identity of the submenu containing the field on which help was requested.

sSubTopic (SHORT)
Subtopic identifier.

control	Control identity of the cursored field and on which help was requested.
---------	---

-1 No menu item was selected

other	Menu item identity of the currently selected submenu item on which help was requested.
-------	--

Action indicator.

Type of window on which help was requested.

HLPM_WINDOW	An application window
HLPM_FRAME	A frame window
HLPM_MENU	A menu window.

Topic identifier.

window	Identity of the window containing the field on which help was requested.
--------	--

Identity of the submenu containing the field on which help was requested.

Subtopic identifier.

control	Control identity of the cursored field and on which help was requested.
---------	---

-1 No menu item was selected

other	Menu item identity of the currently selected submenu item on which help was requested.
-------	--

rc ([BOOL](#))
Action indicator.

HM_HELPSUBITEM_NOT_FOUND - Syntax

The Help Manager sends this message to the application when the user requests help on a field and it cannot find a related entry in the help subtable.

```
param1
    USHORT    usContext    /* Type of window on which help was requested. */

param2
    SHORT     sTopic       /* Topic identifier. */
    SHORT     sSubTopic    /* Subtopic identifier. */
```

HM_HELPSUBITEM_NOT_FOUND - Remarks

If FALSE is returned from this message, the Help Manager displays the extended help window.

The application has the following options:

- Ignore the notification and not display help for that field or window.
 - Display its own window.
 - Use the [HM_DISPLAY_HELP](#) message to tell the Help Manager to display a particular window.
-

HM_HELPSUBITEM_NOT_FOUND - Default Processing

None.

HM_HELPSUBITEM_NOT_FOUND - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

HM_INFORM

HM_INFORM Field - idnum

idnum (USHORT)
Window identity.

The identity that is associated with the hypertext field.

HM_INFORM Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_INFORM Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

0
Reserved value, zero.

HM_INFORM - Parameters

idnum (USHORT)
Window identity.

The identity that is associated with the hypertext field.

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

0
Reserved value, zero.

HM_INFORM - Syntax

This message is used by the Help Manager to notify the application when the user selects a hypertext field that was specified with the **reftype=inform** attribute of the **:link.** tag.

```
param1
    USHORT  idnum      /* Window identity. */

param2
    ULONG   ulReserved /* Reserved value, should be 0. */
```

HM_INFORM - Default Processing

None.

HM_INFORM - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Default Processing](#)
 - [Glossary](#)
-

HM_INVALIDATE_DDF_DATA

HM_INVALIDATE_DDF_DATA Field - rescount

rescount ([ULONG](#))
The count of DDFs to be invalidated.

HM_INVALIDATE_DDF_DATA Field - resarray

resarray ([PUSHORT](#))
Pointer to an array.

The pointer to an array of unsigned 16-bit ([USHORT](#)) integers that are the *res* numbers of DDFs to be invalidated.

Note: If both param1 and param2 are NULL, then all the DDFs in that page will be invalidated.

HM_INVALIDATE_DDF_DATA Return Value - rc

rc ([ULONG](#))

Return code.

0

The procedure was successfully completed.

Other

See the values of the *errorcode* parameter of the HM_ERROR message.

HM_INVALIDATE_DDF_DATA - Parameters

rescount ([ULONG](#))

The count of DDFs to be invalidated.

resarray ([PUSHORT](#))

Pointer to an array.

The pointer to an array of unsigned 16-bit ([USHORT](#)) integers that are the *res* numbers of DDFs to be invalidated.

Note: If both param1 and param2 are NULL, then all the DDFs in that page will be invalidated.

rc ([ULONG](#))

Return code.

0

The procedure was successfully completed.

Other

See the values of the *errorcode* parameter of the HM_ERROR message.

HM_INVALIDATE_DDF_DATA - Syntax

The application sends this message to IPF to indicate that the previous DDF data is no longer valid.

```
param1
    ULONG    rescount /* The count of DDFs to be invalidated. */
param2
    PUSHORT  resarray /* Pointer to an array. */
```

HM_INVALIDATE_DDF_DATA - Remarks

When IPF receives this message, it discards the current DDF data and sends a new HM_QUERY_DDF_DATA message to the object communication window.

This message should be sent to the child of the coverpage window handle.

HM_INVALIDATE_DDF_DATA - Default Processing

None.

HM_INVALIDATE_DDF_DATA - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

HM_KEYS_HELP

HM_KEYS_HELP Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

HM_KEYS_HELP Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

HM_KEYS_HELP Return Value - rc

rc ([ULONG](#))
 Return code.

0
 The keys help window was successfully displayed

Other
 See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_KEYS_HELP - Parameters

ulReserved ([ULONG](#))
 Reserved value, should be 0.

ulReserved ([ULONG](#))
 Reserved value, should be 0.

rc ([ULONG](#))
 Return code.

0
 The keys help window was successfully displayed

Other
 See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_KEYS_HELP - Syntax

This message is sent by the application and informs the help manager to display the keys help window.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

HM_KEYS_HELP - Remarks

When the Help Manager receives this message, it sends a [HM_QUERY_KEYS_HELP](#) message to the active application window. The active application window is the window that was specified when the last [HM_SET_ACTIVE_WINDOW](#) message was sent. If no [HM_SET_ACTIVE_WINDOW](#) message was issued, then the active application window is the window specified in the [WinAssociateHelpInstance](#) call.

The application must return one of the following:

- The identity of a keys help window in the *usHelpPanel* parameter of the [HM_QUERY_KEYS_HELP](#) message.
- Zero, if no action is to be taken by the Help Manager for keys help.

HM_KEYS_HELP - Default Processing

None.

HM_KEYS_HELP - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)
-

HM_LOAD_HELP_TABLE

HM_LOAD_HELP_TABLE Field - idHelpTable

idHelpTable ([USHORT](#))
Identity of the help table.

HM_LOAD_HELP_TABLE Field - fsidentityflag

fsidentityflag ([USHORT](#))
Help table identity indicator.

0xFFFF Reserved value.

HM_LOAD_HELP_TABLE Field - MODULE

MODULE ([HMODULE](#))
Resource identity.

Handle of the module that contains the help table and help subtable.

HM_LOAD_HELP_TABLE Return Value - rc

rc ([ULONG](#))

Return code.

0

The procedure was successfully completed.

Other

See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_LOAD_HELP_TABLE - Parameters

idHelpTable ([USHORT](#))

Identity of the help table.

fsidentityflag ([USHORT](#))

Help table identity indicator.

0xFFFF

Reserved value.

MODULE ([HMODULE](#))

Resource identity.

Handle of the module that contains the help table and help subtable.

rc ([ULONG](#))

Return code.

0

The procedure was successfully completed.

Other

See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_LOAD_HELP_TABLE - Syntax

The application sends this message to give the Help Manager the module handle that contains the help table, the help subtable, and the identity of the help table.

```
param1
    USHORT    idHelpTable    /* Identity of the help table. */
    USHORT    fsidentityflag /* Help table identity indicator. */

param2
    HMODULE   MODULE         /* Resource identity. */
```

HM_LOAD_HELP_TABLE - Default Processing

None.

HM_LOAD_HELP_TABLE - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Default Processing](#)

[Glossary](#)

HM_NOTIFY

HM_NOTIFY Field - controlres

controlres ([USHORT](#))

Res number of the control that was selected.

For author-defined push buttons, this is the res number that was specified with the push button tag (:**pbutton.**). For default push buttons, this is the res number defined in the PMHELP.H file.

HM_NOTIFY Field - usReserve

usReserve ([USHORT](#))

Reserved value, should be 0.

Reserved for events other than CONTROL_SELECTED and HELP_REQUESTED.

HM_NOTIFY Field - usevent

usevent (USHORT)

The type of event which has occurred.

CONTROL_SELECTED	A control was selected.
HELP_REQUESTED	Help was requested.
OPEN_COVERPAGE	The coverpage is displayed.
OPEN_PAGE	The child window of the coverpage is opened.
SWAP_PAGE	The child window of the coverpage is swapped.
OPEN_INDEX	The index window is displayed.
OPEN_TOC	The table of contents window is displayed.
OPEN_HISTORY	The history window is displayed.
OPEN_LIBRARY	The new library is opened.
OPEN_SEARCH_HIT_LIST	The search list displayed.

HM_NOTIFY Field - ulhwnd

ulhwnd (ULONG)

Window handle of relevant window.

HM_NOTIFY Return Value - rc

rc (BOOL)

Return code.

TRUE	
FALSE	IPF will not format the controls and re-size the window.
	IPF will process as normal.

HM_NOTIFY - Parameters

controlres (USHORT)

Res number of the control that was selected.

For author-defined push buttons, this is the res number that was specified with the push button tag (**:pbutton.**). For default push buttons, this is the res number defined in the PMHELP.H file.

usReserve (USHORT)

Reserved value, should be 0.

Reserved for events other than CONTROL_SELECTED and HELP_REQUESTED.

usevent (USHORT)

The type of event which has occurred.

CONTROL_SELECTED	A control was selected.
HELP_REQUESTED	Help was requested.
OPEN_COVERPAGE	The coverpage is displayed.
OPEN_PAGE	The child window of the coverpage is opened.
SWAP_PAGE	The child window of the coverpage is swapped.
OPEN_INDEX	The index window is displayed.
OPEN_TOC	The table of contents window is displayed.
OPEN_HISTORY	The history window is displayed.
OPEN_LIBRARY	The new library is opened.
OPEN_SEARCH_HIT_LIST	The search list displayed.

ulhwnd (ULONG)

Window handle of relevant window.

rc (BOOL)

Return code.

TRUE	IPF will not format the controls and re-size the window.
FALSE	IPF will process as normal.

HM_NOTIFY - Syntax

This message is used by the application to sub-class and change the behavior or appearance of the help window.

```
param1
    USHORT  controlres /* Res number of the control that was selected. */
    USHORT  usReserve  /* Reserved value, should be 0. */
    USHORT  usevent    /* The type of event which has occurred. */

param2
    ULONG   ulhwnd     /* Window handle of relevant window. */
```

HM_NOTIFY - Remarks

This message is sent to the application to notify it of events that the application would be interested in controlling.

HM_NOTIFY - Default Processing

None.

HM_NOTIFY - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

HM_QUERY

HM_QUERY Field - usselectionid

usselectionid (USHORT)	
What is being requested.	
This parameter should be specified only if the query is for HMQW_VIEWPORT and should otherwise be coded as NULL.	
Specifies whether a res ID, ID number, or group number is being requested. The value can be any of the following constants:	
HMQVP_NUMBER	A pointer to a USHORT that holds the res ID of the window.
HMQVP_NAME	A pointer to a null-terminated string that holds the ID of the window.
HMQVP_GROUP	The group number of the window.

HM_QUERY Field - usmessageid

usmessageid (USHORT)	
Type of window queried.	
Specifies the type of window to query. The value can be any of the following constants:	
HMQW_INDEX	The handle of the index window.
HMQW_TOC	The handle of the Table of Contents window.
HMQW_SEARCH	

HMQW_VIEWEDPAGES	The handle of the Search Hitlist window.
HMQW_LIBRARY	The handle of the Viewed Pages window.
HMQW_OBJCOM_WINDOW	The handle of the Library List window.
HMQW_INSTANCE	The handle of the active communication window.
HMQW_COVERPAGE	The handle of the help instance.
	The handle of the help manager multiple document interface (MDI) parent window. It is where the secondary windows are contained within the parent window.
HMQW_VIEWPORT	The handle of the viewport window specified in the low-order word of <i>param1</i> and in <i>param2</i> .
	When HMQW_VIEWPORT is specified in <i>usmessageid</i> , a value must be specified in <i>usselectionid</i> to indicate whether a res ID, ID number, or group number is being requested.
HMQW_GROUP_VIEWPORT	The group number of the window whose handle is specified in <i>param2</i> .
HMQW_RES_VIEWPORT	The res number of the window whose handle is specified in <i>param2</i> .
HMQW_ACTIVEVIEWPORT	The handle of the currently active window.
USERDATA	The previously stored user-data.

HM_QUERY Field - pvoid

pvoid (PVOID)

Varies, depending on value selected above.

param2 depends on the value of *param1 usmessageid*.

If *param1 usmessageid* is HMQW_VIEWPORT, then *param2* is a pointer to the res number, ID, or group ID.

If *param1 usmessageid* is HMQW_GROUP_VIEWPORT, then *param2* is the handle of the viewport for which the group number is assigned.

If *param1 usmessageid* is HMQW_RES_VIEWPORT, then *param2* is the handle of the viewport for which the res number is requested.

HM_QUERY Return Value - rc

rc (ULONG)

Return code.

0
The procedure was not successfully completed.

Other
The handle (HWND), group number (USHORT), or res number (USHORT) of the window, or the user data

(USHORT), depending on the value of *param1 usselectionid*.

HM_QUERY - Parameters

usselectionid (USHORT)

What is being requested.

This parameter should be specified only if the query is for HMQW_VIEWPORT and should otherwise be coded as NULL.

Specifies whether a res ID, ID number, or group number is being requested. The value can be any of the following constants:

HMQVP_NUMBER

A pointer to a USHORT that holds the res ID of the window.

HMQVP_NAME

A pointer to a null-terminated string that holds the ID of the window.

HMQVP_GROUP

The group number of the window.

usmessageid (USHORT)

Type of window queried.

Specifies the type of window to query. The value can be any of the following constants:

HMQW_INDEX

The handle of the index window.

HMQW_TOC

The handle of the Table of Contents window.

HMQW_SEARCH

The handle of the Search Hitlist window.

HMQW_VIEWEDPAGES

The handle of the Viewed Pages window.

HMQW_LIBRARY

The handle of the Library List window.

HMQW_OBJCOM_WINDOW

The handle of the active communication window.

HMQW_INSTANCE

The handle of the help instance.

HMQW_COVERPAGE

The handle of the help manager multiple document interface (MDI) parent window. It is where the secondary windows are contained within the parent window.

HMQW_VIEWPORT

The handle of the viewport window specified in the low-order word of param1 and in param2.

When HMQW_VIEWPORT is specified in *usmessageid*, a value must be specified in *usselectionid* to indicate whether a res ID, ID number, or group number is being requested.

HMQW_GROUP_VIEWPORT

The group number of the window whose handle is specified in param2.

HMQW_RES_VIEWPORT

The res number of the window whose handle is specified in param2.

HMQW_ACTIVEVIEWPORT

The handle of the currently active window.

USERDATA

The previously stored user-data.

pvoid (PVOID)

Varies, depending on value selected above.

param2 depends on the value of *param1 usmessageid*.

If *param1 usmessageid* is HMQW_VIEWPORT, then *param2* is a pointer to the res number, ID, or group ID.

If *param1 usmessageid* is HMQW_GROUP_VIEWPORT, then *param2* is the handle of the viewport for which the group number is assigned.

If *param1 usmessageid* is HMQW_RES_VIEWPORT, then *param2* is the handle of the viewport for which the res number is requested.

rc (ULONG)

Return code.

0

The procedure was not successfully completed.

Other

The handle (HWND), group number (USHORT), or res number (USHORT) of the window, or the user data (USHORT), depending on the value of *param1 usselectionid*.

HM_QUERY - Syntax

This message is sent to IPF by the application to request IPF-specific information, such as the current Instance handle, the active communication object window, the active window, or the group number of the current window.

```
param1
    USHORT  usselectionid /* What is being requested. */
    USHORT  usmessageid  /* Type of window queried. */

param2
    PVOID   pvoid        /* Varies, depending on value selected above. */
```

HM_QUERY - Default Processing

None.

HM_QUERY - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Default Processing](#)

[Glossary](#)

HM_QUERY_DDF_DATA

HM_QUERY_DDF_DATA Field - pageclienthwnd

pageclienthwnd (HWND)

Client handle.

The client handle of the page that contains the object communication window.

HM_QUERY_DDF_DATA Field - resid

resid (ULONG)

The res ID associated with the DDF tag.

HM_QUERY_DDF_DATA Return Value - rc

rc (HDDF)

Return code.

0

An error has occurred in the application's DDF processing.

Other

The DDF handle to be displayed.

Note: Once this handle has been returned, the HDDF handle can no longer be used by the application.

HM_QUERY_DDF_DATA - Parameters

pageclienthwnd (HWND)

Client handle.

The client handle of the page that contains the object communication window.

resid (ULONG)

The res ID associated with the DDF tag.

rc (HDDF)

Return code.

0

An error has occurred in the application's DDF processing.

Other

The DDF handle to be displayed.

Note: Once this handle has been returned, the HDDF handle can no longer be used by the application.

HM_QUERY_DDF_DATA - Syntax

This message is sent to the communication object window by IPF when it encounters the dynamic data formatting (:ddf.) tag.

```
param1
    HWND    pageclienthwnd /* Client handle. */

param2
    ULONG    resid          /* The res ID associated with the DDF tag. */
```

HM_QUERY_DDF_DATA - Remarks

Upon receiving this message, the communication object calls DdfInitialize to indicate the start of dynamic data formatting (DDF). Any combination of other DDF calls are then made to describe this data. When this is complete, the communication object finishes processing this message, indicating that the DDF data is complete. After that time, the DDF handle received from DdfInitialize is considered invalid.

HM_QUERY_DDF_DATA - Default Processing

None.

HM_QUERY_DDF_DATA - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

HM_QUERY_KEYS_HELP

HM_QUERY_KEYS_HELP Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_QUERY_KEYS_HELP Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_QUERY_KEYS_HELP Return Value - usHelpPanel

usHelpPanel (USHORT)
Help panel ID.

The identity of the application-defined keys help window to be displayed.

0	Do nothing
Other	Identity of the keys help window to be displayed.

HM_QUERY_KEYS_HELP - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

usHelpPanel (USHORT)
Help panel ID.

The identity of the application-defined keys help window to be displayed.

0	Do nothing
Other	Identity of the keys help window to be displayed.

HM_QUERY_KEYS_HELP - Syntax

When the user requests the keys help function, the Help Manager sends this message to the application.

```
param1
    ULONG    ulReserved    /* Reserved value, should be 0. */

param2
    ULONG    ulReserved    /* Reserved value, should be 0. */
```

HM_QUERY_KEYS_HELP - Remarks

The application responds by returning the identity of the requested keys help window. The Help Manager then displays that help window. Returning 0 in the *usHelpPanel* parameter indicates that the Help Manager should do nothing for the keys help function.

HM_QUERY_KEYS_HELP - Default Processing

None.

HM_QUERY_KEYS_HELP - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

HM_REPLACE_HELP_FOR_HELP

HM_REPLACE_HELP_FOR_HELP Field - idHelpForHelpPanel

idHelpForHelpPanel ([USHORT](#))
Identity of the application-defined Help for Help window.

0	Use the Help Manager Help for Help window.
Other	Identity of the application-defined Help for Help window.

HM_REPLACE_HELP_FOR_HELP Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_REPLACE_HELP_FOR_HELP Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_REPLACE_HELP_FOR_HELP - Parameters

idHelpForHelpPanel (USHORT)
Identity of the application-defined Help for Help window.

0	Use the Help Manager Help for Help window.
Other	Identity of the application-defined Help for Help window.

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

HM_REPLACE_HELP_FOR_HELP - Syntax

This message tells the Help Manager to display the application-defined Help for Help window instead of the Help Manager Help for Help window.

```
param1
    USHORT idHelpForHelpPanel /* Identity of the application-defined Help for Help window. */
param2
```

ULONG ulReserved /* Reserved value, should be 0. */

HM_REPLACE_HELP_FOR_HELP - Remarks

An application may prefer to provide information that is more specific to itself, rather than the more general help information provided in the Help Manager Help for Help window.

HM_REPLACE_HELP_FOR_HELP - Default Processing

None.

HM_REPLACE_HELP_FOR_HELP - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

HM_REPLACE_USING_HELP

HM_REPLACE_USING_HELP Field - idUsingHelpPanel

idUsingHelpPanel ([USHORT](#))
The identity of the application-defined Using Help window.

0	Use the Help Manager Using Help window.
Other	The identity of the application-defined Using Help window.

HM_REPLACE_USING_HELP Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_REPLACE_USING_HELP Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_REPLACE_USING_HELP - Parameters

idUsingHelpPanel (USHORT)
The identity of the application-defined Using Help window.

0
Use the Help Manager Using Help window.

Other
The identity of the application-defined Using Help window.

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

HM_REPLACE_USING_HELP - Syntax

This message tells the Help Manager to display the application-defined Using help window instead of the Help Manager Using help window.

```
param1
    USHORT    idUsingHelpPanel

param2
    ULONG     ulReserved        /* Reserved value, should be 0. */
```

HM_REPLACE_USING_HELP - Remarks

An application may prefer to provide information that is more specific to itself, rather than the more general help information that is provided in the Help Manager Using help window. The guidelines that define the current CUA interface recommend the **Using help** choice be provided in a pull-down menu from the **Help** choice.

HM_REPLACE_USING_HELP - Default Processing

None.

HM_REPLACE_USING_HELP - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

HM_SET_ACTIVE_WINDOW

HM_SET_ACTIVE_WINDOW Field - hwndActiveWindow

hwndActiveWindow ([HWND](#))

The handle of the window to be made active.

Its window procedure receives all messages from the Help Manager until the application changes the active window with another HM_SET_ACTIVE_WINDOW message.

HM_SET_ACTIVE_WINDOW Field - hwndRelativeWindow

hwndRelativeWindow ([HWND](#))

The handle of the window next to which the help window is to be positioned.

The handle of the application window next to which the Help Manager will position a new help window.

HWND_PARENT

This Help Manager defined constant tells the Help Manager to trace the parent chain of the window that had the focus when the user requested help.

Other

Handle of the window next to which the help window is to be positioned.

HM_SET_ACTIVE_WINDOW Return Value - rc

rc ([ULONG](#))

Return code.

0

The procedure has been successfully completed.

Other

See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_SET_ACTIVE_WINDOW - Parameters

hwndActiveWindow ([HWND](#))

The handle of the window to be made active.

Its window procedure receives all messages from the Help Manager until the application changes the active window with another HM_SET_ACTIVE_WINDOW message.

hwndRelativeWindow ([HWND](#))

The handle of the window next to which the help window is to be positioned.

The handle of the application window next to which the Help Manager will position a new help window.

HWND_PARENT

This Help Manager defined constant tells the Help Manager to trace the parent chain of the window that had the focus when the user requested help.

Other

Handle of the window next to which the help window is to be positioned.

rc ([ULONG](#))

Return code.

0

The procedure has been successfully completed.

Other

See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_SET_ACTIVE_WINDOW - Syntax

This message allows the application to change the window with which the Help Manager communicates and the window to which the help window is to be positioned.

```
param1
    HWND    hwndActiveWindow    /* The handle of the window to be made active. */
```

```
param2
    HWND    hwndRelativeWindow    /* The handle of the window next to which the help window is to be positioned.
```

HM_SET_ACTIVE_WINDOW - Remarks

Normally the Help Manager communicates with the application window with which the Help Manager instance has been associated. The help window is positioned next to this same application window.

If the *hwndActiveWindow* parameter is 0, the *hwndRelativeWindow* parameter is set to 0. That is, if the active window is NULL HANDLE, the relative window is not used.

HM_SET_ACTIVE_WINDOW - Default Processing

None.

HM_SET_ACTIVE_WINDOW - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

HM_SET_COVERPAGE_SIZE

HM_SET_COVERPAGE_SIZE Field - coverpagerectl

coverpagerectl ([PRECTL](#))

Pointer to [RECTL](#) containing the size of the coverpage.

HM_SET_COVERPAGE_SIZE Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

HM_SET_COVERPAGE_SIZE Return Value - rc

rc ([ULONG](#))

Return code.

0

The procedure was successfully completed.

Other

See the values of the *errorcode* parameter of the HM_ERROR message.

HM_SET_COVERPAGE_SIZE - Parameters

coverpagerectl ([PRECTL](#))

Pointer to [RECTL](#) containing the size of the coverpage.

ulReserved ([ULONG](#))

Reserved value, should be 0.

rc ([ULONG](#))

Return code.

0

The procedure was successfully completed.

Other

See the values of the *errorcode* parameter of the HM_ERROR message.

HM_SET_COVERPAGE_SIZE - Syntax

This message is sent to IPF by the application to set the size of the coverpage, the window within which all other IPF windows are displayed.

```
param1
    PRECTL    coverpagerectl    /* Pointer to RECTL containing the size of the coverpage. */

param2
    ULONG    ulReserved        /* Reserved value, should be 0. */
```

HM_SET_COVERPAGE_SIZE - Remarks

The default size for the coverpage of a book is the full width of the screen, while the default size for a help file is one-half the width of the screen.

This message takes effect immediately, changing the size of the coverpage. If the coverpage is not currently open, the requested size is saved for the next open.

HM_SET_COVERPAGE_SIZE - Default Processing

None.

HM_SET_COVERPAGE_SIZE - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

HM_SET_HELP_LIBRARY_NAME

HM_SET_HELP_LIBRARY_NAME Field - pszHelpLibraryName

pszHelpLibraryName ([PSZ](#))

Library name.

This points to a string that contains a list of help window library names that will be searched by the Help Manager for the requested help window. The names must be separated by a blank.

HM_SET_HELP_LIBRARY_NAME Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

HM_SET_HELP_LIBRARY_NAME Return Value - rc

rc ([ULONG](#))

Return code.

0

The newly specified library successfully replaced the current help window library name.

Other

See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_SET_HELP_LIBRARY_NAME - Parameters

pszHelpLibraryName ([PSZ](#))

Library name.

This points to a string that contains a list of help window library names that will be searched by the Help Manager for the requested help window. The names must be separated by a blank.

ulReserved ([ULONG](#))

Reserved value, should be 0.

rc ([ULONG](#))

Return code.

0

The newly specified library successfully replaced the current help window library name.

Other

See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_SET_HELP_LIBRARY_NAME - Syntax

This message identifies a list of help window library names to the Help Manager instance.

```
param1
    PSZ    pszHelpLibraryName /* Library name. */

param2
    ULONG  ulReserved          /* Reserved value, should be 0. */
```

HM_SET_HELP_LIBRARY_NAME - Remarks

Any subsequent communication to the Help Manager with this message replaces the current list of names with the newly specified list.

When help is requested, the Help Manager will search each library in the list for the requested help window.

HM_SET_HELP_LIBRARY_NAME - Default Processing

None.

HM_SET_HELP_LIBRARY_NAME - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

HM_SET_HELP_WINDOW_TITLE

HM_SET_HELP_WINDOW_TITLE Field - pszHelpWindowTitle

pszHelpWindowTitle ([PSZ](#))
Help window title.

This points to a string containing the new Help Window title.

HM_SET_HELP_WINDOW_TITLE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

HM_SET_HELP_WINDOW_TITLE Return Value - rc

- rc** ([ULONG](#))
Return code.
- | | |
|-------|---|
| 0 | The window title was successfully set. |
| Other | See the values of the <i>ulErrorCode</i> parameter of the HM_ERROR message. |

HM_SET_HELP_WINDOW_TITLE - Parameters

pszHelpWindowTitle ([PSZ](#))

Help window title.

This points to a string containing the new Help Window title.

ulReserved ([ULONG](#))

Reserved value, should be 0.

rc ([ULONG](#))

Return code.

0

The window title was successfully set.

Other

See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_SET_HELP_WINDOW_TITLE - Syntax

This message allows the application to change the window text of a help window title.

```
param1
    PSZ    pszHelpWindowTitle /* Help window title. */

param2
    ULONG  ulReserved         /* Reserved value, should be 0. */
```

HM_SET_HELP_WINDOW_TITLE - Default Processing

None.

HM_SET_HELP_WINDOW_TITLE - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Default Processing](#)

[Glossary](#)

HM_SET_OBJCOM_WINDOW

HM_SET_OBJCOM_WINDOW Field - objcomhwnd

objcomhwnd ([HWND](#))

Handle of the communication object window to be set.

HM_SET_OBJCOM_WINDOW Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

HM_SET_OBJCOM_WINDOW Return Value - hwndprevioushwnd

hwndprevioushwnd ([HWND](#))

The handle of the previous communication object window.

HM_SET_OBJCOM_WINDOW - Parameters

objcomhwnd ([HWND](#))

Handle of the communication object window to be set.

ulReserved ([ULONG](#))

Reserved value, should be 0.

hwndprevioushwnd ([HWND](#))

The handle of the previous communication object window.

HM_SET_OBJCOM_WINDOW - Syntax

This message is sent to IPF by the application to identify the communication object window to which the HM_INFORM and

HM_QUERY_DDF_DATA messages will be sent. This message is not necessary if the communication object does not expect to receive either of these messages.

```
hwndparam1
    HWND    objcomhwnd    /* Handle of the communication object window to be set. */

param2
    ULONG    ulReserved    /* Reserved value, should be 0. */
```

HM_SET_OBJCOM_WINDOW - Remarks

HM_INFORM and HM_QUERY_DDF_DATA messages which are not processed must be passed to the previous communication object window which was returned when HM_SET_OBJECT_WINDOW was sent.

HM_SET_OBJCOM_WINDOW - Default Processing

None.

HM_SET_OBJCOM_WINDOW - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

HM_SET_SHOW_PANEL_ID

HM_SET_SHOW_PANEL_ID Field - fsShowPanelId

fsShowPanelId ([USHORT](#))
The show window identity indicator.

CMIC_HIDE_PANEL_ID
Sets the show option off and the window identity is not displayed.

CMIC_SHOW_PANEL_ID
Sets the show option on and the window identity is displayed.

CMIC_TOGGLE_PANEL_ID
Toggles the display of the window identity.

HM_SET_SHOW_PANEL_ID Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_SET_SHOW_PANEL_ID Parameter - rc

rc (ULONG)
Return code.

0
The show window identity indicator was successfully changed.

Other
See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_SET_SHOW_PANEL_ID - Parameters

fsShowPanelId (USHORT)
The show window identity indicator.

CMIC_HIDE_PANEL_ID
Sets the show option off and the window identity is not displayed.

CMIC_SHOW_PANEL_ID
Sets the show option on and the window identity is displayed.

CMIC_TOGGLE_PANEL_ID
Toggles the display of the window identity.

ulReserved (ULONG)
Reserved value, should be 0.

rc (ULONG)
Return code.

0
The show window identity indicator was successfully changed.

Other
See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_SET_SHOW_PANEL_ID - Syntax

This message tells the Help Manager to display, hide, or toggle the window identity for each help window displayed.

```
param1
    USHORT  fsShowPanelId  /* The show window identity indicator. */

param2
    ULONG   ulReserved     /* Reserved value, should be 0. */
```

HM_SET_SHOW_PANEL_ID - Default Processing

None.

HM_SET_SHOW_PANEL_ID - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Default Processing](#)
 - [Glossary](#)

HM_SET_USERDATA

HM_SET_USERDATA Field - ulReserved

ulReserved (**ULONG**)
Reserved value, should be 0.

HM_SET_USERDATA Field - usrdata

usrdata (**VOID**)
4-byte user data area.

HM_SET_USERDATA Parameter - rc

rc (**ULONG**)
Return code.

TRUE	The user data was successfully stored.
FALSE	The call failed.

HM_SET_USERDATA - Parameters

ulReserved (**ULONG**)
Reserved value, should be 0.

usrdata (**VOID**)
4-byte user data area.

rc (**ULONG**)
Return code.

TRUE	The user data was successfully stored.
FALSE	The call failed.

HM_SET_USERDATA - Syntax

The application sends this message to IPF to store data in the IPF data area.

```
param1
    ULONG  ulReserved  /*  Reserved value, should be 0. */
param2
    VOID    usrdata    /*  4-byte user data area. */
```

HM_SET_USERDATA - Default Processing

None.

HM_SET_USERDATA - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Default Processing](#)
[Glossary](#)

HM_TUTORIAL

HM_TUTORIAL Field - pszTutorialName

pszTutorialName ([PSZ](#))
Default tutorial name.

This points to a string that contains the name of the default tutorial program specified in the Help Manager initialization structure. A tutorial name specified in the help window definition overrides this default tutorial program.

HM_TUTORIAL Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

HM_TUTORIAL Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

HM_TUTORIAL - Parameters

pszTutorialName ([PSZ](#))
Default tutorial name.

This points to a string that contains the name of the default tutorial program specified in the Help Manager initialization structure. A tutorial name specified in the help window definition overrides this default tutorial program.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

HM_TUTORIAL - Syntax

The Help Manager sends this message to the application window when the user selects the Tutorial choice from a help window.

```
param1
    PSZ    pszTutorialName /* Default tutorial name. */

param2
    ULONG  ulReserved      /* Reserved value, should be 0. */
```

HM_TUTORIAL - Remarks

The application then calls its own tutorial program.

HM_TUTORIAL - Default Processing

None.

HM_TUTORIAL - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

HM_UPDATE_OBJCOM_WINDOW_CHAIN

HM_UPDATE_OBJCOM_WINDOW_CHAIN Field - hwnd

hwnd (HWND)
The handle of the object to be withdrawn from the communication chain.

HM_UPDATE_OBJCOM_WINDOW_CHAIN Field - hwnd

hwnd (HWND)
Window containing the handle of the object to be replaced.

HM_UPDATE_OBJCOM_WINDOW_CHAIN Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_UPDATE_OBJCOM_WINDOW_CHAIN - Parameters

hwnd (HWND)
The handle of the object to be withdrawn from the communication chain.

hwnd (HWND)
Window containing the handle of the object to be replaced.

ulReserved (ULONG)
Reserved value, should be 0.

HM_UPDATE_OBJCOM_WINDOW_CHAIN - Syntax

This message is sent to the currently active communication object by the communication object who wants to withdraw from the communication chain.

```
param1      HWND    hwnd      /* The handle of the object to be withdrawn from the communication chain. */
param2      HWND    hwnd      /* Window containing the handle of the object to be replaced. */
```

HM_UPDATE_OBJCOM_WINDOW_CHAIN - Remarks

The object that receives this message should check to see if the object handle returned from [HM_SET_OBJCOM_WINDOW](#) is equal to the handle in *param1*. If the handle is equal, then the handle in *param1* should be replaced by the handle in *param2*. If the handle is not equal *and* the handle previously received is not NULL HANDLE, then send HM_UPDATE_OBJCOM_WINDOW_CHAIN to that object.

HM_UPDATE_OBJCOM_WINDOW_CHAIN - Default Processing

None.

HM_UPDATE_OBJCOM_WINDOW_CHAIN - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

Language Support Dialog Processing

This system-provided window procedure processes messages for a dialog that has been created or loaded specifying a 'NULL' dialog procedure.

For any other messages the Language Support Dialog Procedure issues and returns the result of the [WinDefDlgProc](#) function.

Language Support Dialog Messages

This section describes the actions taken by the Language Support Dialog Procedure when the following messages are received.

WM_ACTIVATE (Language Support Dialog)

WM_ACTIVATE (Language Support Dialog) - Syntax

For the cause of this message, see [WM_ACTIVATE](#).

For a description of the parameters, see [WM_ACTIVATE](#).

WM_ACTIVATE (Language Support Dialog) - Remarks

The Language Support Dialog Procedure responds to this message by issuing the [WinDefDlgProc](#) function, then posting a [WM_PACTIVATE](#) message to the application queue and setting *uReserved* to the result of the [WinDefDlgProc](#) function.

WM_ACTIVATE (Language Support Dialog) - Default Processing

The default window procedure takes no action on this message, other than to set *uReserved* to 0.

WM_ACTIVATE (Language Support Dialog) - Related Messages

Related Messages

- [WM_ACTIVATE](#)
-

WM_ACTIVATE (Language Support Dialog) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_CONTROL (Language Support Dialog)

WM_CONTROL (Language Support Dialog) - Syntax

For the cause of this message, see [WM_CONTROL](#).

For a description of the parameters, see [WM_CONTROL](#).

WM_CONTROL (Language Support Dialog) - Remarks

The Language Support Dialog Procedure responds to this message by issuing the [WinDefDlgProc](#) function, then posting a [WM_PCONTROL](#) message to the application queue and setting *uReserved* to the result of the [WinDefDlgProc](#) function.

WM_CONTROL (Language Support Dialog) - Default Processing

The default window procedure takes no action on this message, other than to set *uReserved* to 0.

WM_CONTROL (Language Support Dialog) - Related Messages

Related Messages

- [WM_CONTROL](#)
-

WM_CONTROL (Language Support Dialog) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_PAINT (Language Support Dialog)

WM_PAINT (Language Support Dialog) - Syntax

For the cause of this message, see [WM_PAINT](#).

For a description of the parameters, see [WM_PAINT](#).

WM_PAINT (Language Support Dialog) - Remarks

The Language Support Dialog Procedure responds to this message by issuing the [WinDefDlgProc](#) function, then posting a [WM_PPAINT](#) message to the application queue and setting *uReserved* to the result of the [WinDefDlgProc](#) function.

The [WinBeginPaint](#) and [WinEndPaint](#) functions are issued by the Language Support Dialog Procedure, during the processing of the [WM_PPAINT](#) message.

WM_PAINT (Language Support Dialog) - Default Processing

The default window procedure issues the [WinBeginPaint](#) and [WinEndPaint](#) functions, and then sets *uReserved* to 0.

WM_PAINT (Language Support Dialog) - Related Messages

Related Messages

- [WM_PAINT](#)
-

WM_PAINT (Language Support Dialog) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_PPAINT (Language Support Dialog)

WM_PPAINT (Language Support Dialog) - Syntax

For the cause of this message, see [WM_PAINT](#).

For a description of the parameters, see [WM_PAINT](#).

WM_PAINT (Language Support Dialog) - Remarks

The Language Support Dialog Procedure issuing the [WinDefDlgProc](#) function, then issues the [WinBeginPaint](#) and [WinEndPaint](#) functions, and then setting *ulReserved* to the result of the [WinDefDlgProc](#) function.

WM_PAINT (Language Support Dialog) - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_PAINT (Language Support Dialog) - Related Messages

Related Messages

- [WM_PAINT](#)

WM_PAINT (Language Support Dialog) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_SETFOCUS (Language Support Dialog)

WM_SETFOCUS (Language Support Dialog) - Syntax

For the cause of this message, see [WM_SETFOCUS](#).

For a description of the parameters, see [WM_SETFOCUS](#).

WM_SETFOCUS (Language Support Dialog) - Remarks

The Language Support Dialog Procedure responds to this message by issuing the [WinDefDlgProc](#) function, then posting a [WM_PSETFOCUS](#) message to the application queue and setting *uReserved* to the result of the [WinDefDlgProc](#) function.

WM_SETFOCUS (Language Support Dialog) - Default Processing

The default window procedure takes no action on this message, other than to set *uReserved* to 0.

WM_SETFOCUS (Language Support Dialog) - Related Messages

Related Messages

- [WM_SETFOCUS](#)
-

WM_SETFOCUS (Language Support Dialog) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_SIZE (Language Support Dialog)

WM_SIZE (Language Support Dialog) - Syntax

For the cause of this message, see [WM_SIZE](#).

For a description of the parameters, see [WM_SIZE](#).

WM_SIZE (Language Support Dialog) - Remarks

The Language Support Dialog Procedure responds to this message by issuing the [WinDefDlgProc](#) function, then posting a [WM_PSIZE](#) message to the application queue and setting *ulReserved* to the result of the [WinDefDlgProc](#) function.

WM_SIZE (Language Support Dialog) - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_SIZE (Language Support Dialog) - Related Messages

Related Messages

- [WM_SIZE](#)
-

WM_SIZE (Language Support Dialog) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_SYSCOLORCHANGE (Language Support Dialog)

WM_SYSCOLORCHANGE (Language Support Dialog) - Syntax

For the cause of this message, see [WM_SYSCOLORCHANGE](#).

For a description of the parameters, see [WM_SYSCOLORCHANGE](#).

WM_SYSCOLORCHANGE (Language Support Dialog) -

Remarks

The Language Support Dialog Procedure responds to this message by issuing the [WinDefDlgProc](#) function, then posting a [WM_SYSCOLORCHANGE](#) message to the application queue and setting *ulReserved* to the result of the [WinDefDlgProc](#) function.

WM_SYSCOLORCHANGE (Language Support Dialog) - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_SYSCOLORCHANGE (Language Support Dialog) - Related Messages

Related Messages

- [WM_SYSCOLORCHANGE](#)
-

WM_SYSCOLORCHANGE (Language Support Dialog) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

Language Support Window Processing

This system-provided window procedure processes messages for a window that has been created with a window class specifying a "NULL" window procedure.

This section describes the WM_* messages and the language support window procedure action.

For any other messages the Language Support Window Procedure performs the same actions as the Default Window Procedure.

Language Support Window Messages

This section describes the actions taken by the Language Support Window Procedure when the following messages are received.

WM_ACTIVATE (Language Support Window)

WM_ACTIVATE (Language Support Window) - Syntax

For the cause of this message, see [WM_ACTIVATE](#).

For a description of the parameters, see [WM_ACTIVATE](#).

WM_ACTIVATE (Language Support Window) - Remarks

The Language Support Window Procedure responds to this message by posting a [WM_PACTIVATE](#) message to the application queue and setting *uIfReserved* to 0.

WM_ACTIVATE (Language Support Window) - Default Processing

The default window procedure takes no action on this message, other than to set *uIfReserved* to 0.

WM_ACTIVATE (Language Support Window) - Related Messages

Related Messages

- [WM_ACTIVATE](#)
-

WM_ACTIVATE (Language Support Window) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_CONTROL (Language Support Window)

WM_CONTROL (Language Support Window) - Syntax

For the cause of this message, see [WM_CONTROL](#).

For a description of the parameters, see [WM_CONTROL](#).

WM_CONTROL (Language Support Window) - Remarks

The Language Support Window Procedure responds to this message by posting a [WM_PCONTROL](#) message to the application queue and setting *uiReserved* to 0.

WM_CONTROL (Language Support Window) - Default Processing

The default window procedure takes no action on this message, other than to set *uiReserved* to 0.

WM_CONTROL (Language Support Window) - Related Messages

Related Messages

- [WM_CONTROL](#)
-

WM_CONTROL (Language Support Window) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_PAINT (Language Support Window)

WM_PAINT (Language Support Window) - Syntax

For the cause of this message, see [WM_PAINT](#).

For a description of the parameters, see [WM_PAINT](#).

WM_PAINT (Language Support Window) - Remarks

The Language Support Window Procedure responds to this message by posting a [WM_PPAINT](#) message to the application queue and setting *uReserved* to 0.

The [WinBeginPaint](#) and [WinEndPaint](#) functions are issued by the Language Support Window Procedure, during the processing of the [WM_PPAINT](#) message.

WM_PAINT (Language Support Window) - Default Processing

The default window procedure issues the [WinBeginPaint](#) and [WinEndPaint](#) functions, and then sets *uReserved* to 0.

WM_PAINT (Language Support Window) - Related Messages

Related Messages

- [WM_PAINT](#)
-

WM_PAINT (Language Support Window) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_PPAINT (Language Support Window)

WM_PPAINT (Language Support Window) - Syntax

For the cause of this message, see [WM_PPAINT](#).

For a description of the parameters, see [WM_PPAINT](#).

WM_PPAINT (Language Support Window) - Remarks

The Language Support Window Procedure issues the [WinBeginPaint](#) and [WinEndPaint](#) functions, and then sets *ulReserved* to 0.

WM_PPAINT (Language Support Window) - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_PPAINT (Language Support Window) - Related Messages

Related Messages

- [WM_PPAINT](#)
-

WM_PPAINT (Language Support Window) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_SETFOCUS (Language Support Window)

WM_SETFOCUS (Language Support Window) - Syntax

For the cause of this message, see [WM_SETFOCUS](#).

For a description of the parameters, see [WM_SETFOCUS](#).

WM_SETFOCUS (Language Support Window) - Remarks

The Language Support Window Procedure responds to this message by posting a [WM_PSETFOCUS](#) message to the application queue and setting *ulReserved* to 0.

WM_SETFOCUS (Language Support Window) - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_SETFOCUS (Language Support Window) - Related Messages

Related Messages

- [WM_SETFOCUS](#)
-

WM_SETFOCUS (Language Support Window) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_SIZE (Language Support Window)

WM_SIZE (Language Support Window) - Syntax

For the cause of this message, see [WM_SIZE](#).

For a description of the parameters, see [WM_SIZE](#).

WM_SIZE (Language Support Window) - Remarks

The Language Support Window Procedure responds to this message by posting a [WM_PSIZE](#) message to the application queue and setting *ulReserved* to 0.

WM_SIZE (Language Support Window) - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_SIZE (Language Support Window) - Related Messages

Related Messages

- [WM_SIZE](#)
-

WM_SIZE (Language Support Window) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_SYSCOLORCHANGE (Language Support Window)

WM_SYSCOLORCHANGE (Language Support Window) -

Syntax

For the cause of this message, see [WM_SYSCOLORCHANGE](#).

For a description of the parameters, see [WM_SYSCOLORCHANGE](#).

WM_SYSCOLORCHANGE (Language Support Window) - Remarks

The Language Support Window Procedure responds to this message by posting a [WM_PSYSOLORCHANGE](#) message to the application queue and setting *ulReserved* to 0.

WM_SYSCOLORCHANGE (Language Support Window) - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_SYSCOLORCHANGE (Language Support Window) - Related Messages

Related Messages

- [WM_SYSCOLORCHANGE](#)
-

WM_SYSCOLORCHANGE (Language Support Window) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

List Box Control Window Processing

This system-provided window procedure processes the actions on a list box control (WC_LISTBOX).

Purpose

A list box control is a window containing a list of items. Each item in a list box contains a text string (0 or more characters) and a handle. The text string is displayed in the list box window. The handle can be used by the application to refer to other data associated with each item.

List Box Control Styles

These list box control styles are available:

LS_HORZSCROLL

The list box control enables the operator to scroll the list box horizontally.

LS_MULTIPLESEL

The list box control enables the operator to select more than one item at any one time. Lists that do not have this style allow only a single selection at any one time. If this style is specified, LS_EXTENDEDSEL should also be specified.

LS_EXTENDEDSEL

If this style is specified, the extended selection user interface is enabled.

LS_OWNERDRAW

The list box control has one or more items that can be drawn by the owner. Typically, these items are represented by bit maps rather than by text strings.

LS_NOADJUSTPOS

If this style is included, the list box control is drawn at the size specified. This can cause parts of an item to be shown.

Default Colors

The following system colors are used when the system draws list-box controls:

SYSCLR_FIELDBACKGROUND
SYSCLR_BUTTONDARK
SYSCLR_WINDOW
SYSCLR_WINDOWTEXT
SYSCLR_ENTRYFIELD
SYSCLR_HILITEFOREGROUND
SYSCLR_HILITEBACKGROUND
SYSCLR_WINDOWFRAME

Some of these defaults can be replaced by using the following presentation parameters in the application resource script file or source code:

PP_DISABLEDFOREGROUND
PP_FOREGROUND
PP_HILITEFOREGROUND
PP_BORDERCOLOR

List Box Control Notification Messages

These messages are initiated by the list box control window to notify its owner of significant events.

WM_CONTROL (in List Boxes)

WM_CONTROL (in List Boxes) Field - id

id ([USHORT](#))
Control-window identity.

WM_CONTROL (in List Boxes) Field - usnotifycode

usnotifycode ([USHORT](#))
Notify code.

The list box control window procedure uses these notification codes:

- LN_ENTER** Either the Enter or Return key has been pressed while the list box control has the focus, or the list box control has been double-clicked.
- LN_KILLFOCUS** The list box control loses the focus.
- LN_SCROLL** The list box control is about to scroll horizontally. This can happen when the application has issued a [WinScrollWindow](#) function.
- LN_SETFOCUS** The list box control receives the focus.
- LN_SELECT** An item is being selected (or deselected).
- Note:** To discover the index of the selected item, the application must use the [LM_QUERYSELECTION](#) message.

WM_CONTROL (in List Boxes) Field - hwndcontrolspect

hwndcontrolspect ([HWND](#))
List box control window handle.

WM_CONTROL (in List Boxes) Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_CONTROL (in List Boxes) - Parameters

id ([USHORT](#))

Control-window identity.

usnotifycode ([USHORT](#))

Notify code.

The list box control window procedure uses these notification codes:

LN_ENTER

Either the Enter or Return key has been pressed while the list box control has the focus, or the list box control has been double-clicked.

LN_KILLFOCUS

The list box control loses the focus.

LN_SCROLL

The list box control is about to scroll horizontally. This can happen when the application has issued a [WinScrollWindow](#) function.

LN_SETFOCUS

The list box control receives the focus.

LN_SELECT

An item is being selected (or deselected).

Note: To discover the index of the selected item, the application must use the [LM_QUERYSELECTION](#) message.

hwndcontrolspect ([HWND](#))

List box control window handle.

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_CONTROL (in List Boxes) - Syntax

For the cause of this message, see [WM_CONTROL](#).

```
param1
    USHORT id           /* Control-window identity. */
    USHORT usnotifycode /* Notify code. */

param2
    HWND hwndcontrolspect /* List box control window handle. */
```

WM_CONTROL (in List Boxes) - Remarks

The list box control window procedure generates this message and sends it to its owner, informing the owner of this event.

WM_CONTROL (in List Boxes) - Default Processing

The default window procedure takes no action on this message, other than to set *uiReserved* to 0.

WM_CONTROL (in List Boxes) - Related Messages

Related Messages

- [WM_CONTROL](#)
-

WM_CONTROL (in List Boxes) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_DRAWITEM (in List Boxes)

WM_DRAWITEM (in List Boxes) Field - idListBox

idListBox ([USHORT](#))

Window identifier.

The window identity of the list box control sending this notification message.

WM_DRAWITEM (in List Boxes) Field - pOwnerItem

pOwnerItem ([OWNERITEM](#))

Owner-item structure.

This points to an owner-item structure; see [OWNERITEM](#).

WM_DRAWITEM (in List Boxes) Return Value - rc

rc ([BOOL](#))

Item-drawn indicator.

TRUE

The owner draws the item, so the list box control does not draw it.

FALSE

If the item contains text and the owner does not draw the item, the owner returns this value, and the list box control draws the item.

WM_DRAWITEM (in List Boxes) - Parameters

idListBox ([USHORT](#))

Window identifier.

The window identity of the list box control sending this notification message.

pOwnerItem ([POWNERITEM](#))

Owner-item structure.

This points to an owner-item structure; see [OWNERITEM](#).

rc ([BOOL](#))

Item-drawn indicator.

TRUE

The owner draws the item, so the list box control does not draw it.

FALSE

If the item contains text and the owner does not draw the item, the owner returns this value, and the list box control draws the item.

WM_DRAWITEM (in List Boxes) - Syntax

This notification is sent to the owner of a list box control each time an item is to be drawn.

```
param1
    USHORT        idListBox    /* Window identifier. */

param2
    POWNERITEM    pOwnerItem  /* Owner-item structure. */
```

WM_DRAWITEM (in List Boxes) - Remarks

The list box control window procedure only draws items that are represented by text strings and emphasizes selected items by inverting

them.

If an application uses list box controls containing items that are not represented by text strings, or requires that the emphasized state of an item is to be drawn in a special manner, the list box control must specify the style `LS_OWNERDRAW` and those items must be drawn by the owner.

The list box control window procedure generates this message and sends it to the owner of the list box control, informing the owner that an item is to be drawn, offering the owner the opportunity to draw that item, and indicating that either the item has been drawn, or that the list box control is to draw it.

The item text must not be changed during the processing of this message.

WM_DRAWITEM (in List Boxes) - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set `rc` to the default value of `FALSE`.

WM_DRAWITEM (in List Boxes) - Related Messages

Related Messages

- [WM_DRAWITEM](#)

WM_DRAWITEM (in List Boxes) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_MEASUREITEM (in List Boxes)

WM_MEASUREITEM (in List Boxes) Field - sListBox

sListBox ([SHORT](#))
List-box identifier.

WM_MEASUREITEM (in List Boxes) Field - sltemIndex

sltemIndex ([SHORT](#))
Item index.

The zero-based index of the item which has changed.

WM_MEASUREITEM (in List Boxes) Field - sHeight

sHeight ([SHORT](#))
Height of item.

WM_MEASUREITEM (in List Boxes) Field - sWidth

sWidth ([SHORT](#))
Width of item.

This value is required only if the list box control is scrollable horizontally, that is, it has a style of LS_HORZSCROLL.

WM_MEASUREITEM (in List Boxes) - Parameters

sListBox ([SHORT](#))
List-box identifier.

sltemIndex ([SHORT](#))
Item index.

The zero-based index of the item which has changed.

sHeight ([SHORT](#))
Height of item.

sWidth ([SHORT](#))
Width of item.

This value is required only if the list box control is scrollable horizontally, that is, it has a style of LS_HORZSCROLL.

WM_MEASUREITEM (in List Boxes) - Syntax

This notification is sent to the owner of a list box control to establish the height and width for an item in that control.

```
param1
    SHORT  sListBox    /* List-box identifier. */

param2
    SHORT  sItemIndex /* Item index. */

returns
    SHORT  sHeight    /* Height of item. */
    SHORT  sWidth     /* Width of item. */
```

WM_MEASUREITEM (in List Boxes) - Remarks

This message is sent to the owner of a list box that has a style of `LS_OWNERDRAW`, to offer the owner an opportunity to establish the height and width (for a horizontally scrollable list box control) of an item that accommodates any special requirements for the drawing of items in that list box. It is sent when items in the list box are inserted or deleted, and also when presentation parameters for the list box change.

All items in a list box must have the same height, which must be greater than or equal to the height of the current font.

In particular, this notification is sent to the owner of a list box that has a style of `LS_OWNERDRAW`, to offer the owner an opportunity to establish the height and width (for a horizontally scrollable list box control) of an item that accommodates any special requirements for the drawing of items in that list box.

WM_MEASUREITEM (in List Boxes) - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sHeight* to the default value of 0.

WM_MEASUREITEM (in List Boxes) - Related Messages

Related Messages

- [WM_MEASUREITEM](#)
-

WM_MEASUREITEM (in List Boxes) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)

List Box Control Window Messages

This section describes the list box control window procedure actions on receiving the following messages.

LM_DELETEALL

LM_DELETEALL Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

LM_DELETEALL Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

LM_DELETEALL Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

LM_DELETEALL - Parameters

ulReserved (ULONG)

Reserved value, should be 0.

ulReserved ([ULONG](#))

Reserved value, should be 0.

rc ([BOOL](#))

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

LM_DELETEALL - Syntax

This message is sent to a list box control to delete all the items in the list box.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */
param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

LM_DELETEALL - Remarks

The list box control window procedure responds to this message by deleting all the items in the list box and by setting *rc* to TRUE.

LM_DELETEALL - Default Processing

The default window procedure does not expect to receive this message and, therefore, takes no action on it, other than to set *rc* to the default value of FALSE.

LM_DELETEALL - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

LM_DELETEITEM

LM_DELETEITEM Field - sltemIndex

sltemIndex ([SHORT](#))

Item index.

The zero-based index of the item to be deleted.

LM_DELETEITEM Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

LM_DELETEITEM Return Value - sltemsLeft

sltemsLeft ([SHORT](#))

Number remaining.

The number of items in the list after the item is deleted.

LM_DELETEITEM - Parameters

sltemIndex ([SHORT](#))

Item index.

The zero-based index of the item to be deleted.

ulReserved ([ULONG](#))

Reserved value, should be 0.

sltemsLeft ([SHORT](#))

Number remaining.

The number of items in the list after the item is deleted.

LM_DELETEITEM - Syntax

This message deletes an item from the list box control.

```
param1
    SHORT  sItemIndex /* Item index. */

param2
    ULONG  ulReserved /* Reserved value, should be 0. */
```

LM_DELETEITEM - Remarks

The list box control window procedure responds to this message by deleting the indexed item of the list box and by setting *sItemsLeft* to the count of the items in the list after the item is deleted.

LM_DELETEITEM - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sItemsLeft* to the default value of 0.

LM_DELETEITEM - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

LM_INSERTITEM

LM_INSERTITEM Field - sltemIndex

```
sltemIndex (SHORT)
    Item index.

LIT_END
```

	Add the item to the end of the list.
LIT_SORTASCENDING	Insert the item into the list sorted in ascending order.
LIT_SORTDESCENDING	Insert the item into the list sorted in descending order.
Other	Insert the item into the list at the offset specified by this zero-based index.

LM_INSERTITEM Field - pszItemText

pszItemText ([PSZ](#))
Item text.

This points to a string containing the item text.

LM_INSERTITEM Return Value - sIndexInserted

sIndexInserted ([SHORT](#))
Index of inserted item.

LIT_MEMERROR	The list box control cannot allocate space to insert the list item in the list.
LIT_ERROR	An error, other than LIT_MEMERROR, occurred.
Other	The zero-based index of the offset of the item within the list.

LM_INSERTITEM - Parameters

sItemIndex ([SHORT](#))
Item index.

LIT_END	Add the item to the end of the list.
LIT_SORTASCENDING	Insert the item into the list sorted in ascending order.
LIT_SORTDESCENDING	Insert the item into the list sorted in descending order.
Other	Insert the item into the list at the offset specified by this zero-based index.

pszItemText ([PSZ](#))
Item text.

This points to a string containing the item text.

sIndexInserted ([SHORT](#))
Index of inserted item.

LIT_MEMERROR	The list box control cannot allocate space to insert the list item in the list.
LIT_ERROR	An error, other than LIT_MEMERROR, occurred.
Other	The zero-based index of the offset of the item within the list.

LM_INSERTITEM - Syntax

This message inserts an item into a list box control.

```
param1
    SHORT  sItemIndex    /* Item index. */

param2
    PSZ    pszItemText   /* Item text. */
```

LM_INSERTITEM - Remarks

The list box control window procedure responds to this message by inserting the item text identified by the *pszItemText* parameter into the position in the list specified by the *sItemIndex* parameter.

The sorting sequence used is that defined by the [WinCompareStrings](#) function.

The list box control sets *sIndexInserted* to the zero-based index of the offset of the item within the list.

LM_INSERTITEM - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sIndexInserted* to the default value of 0.

LM_INSERTITEM - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

LM_INSERTMULTITEMS

LM_INSERTMULTITEMS Field - pListboxInfo

pListboxInfo ([PLBOXINFO](#))

Pointer to a structure containing list box information.

LM_INSERTMULTITEMS Field - ppszText

ppszText ([PSZ *](#))

Pointer to an array of pointers to text strings.

This parameter is a pointer to an array of pointers to zero-terminated strings. The array must contain at least *ulItemCount* items. (*ulItemCount* is a field in [LBOXINFO](#).)

If this parameter is set to NULL, a *ulItemCount* number of empty items are inserted into the list. This is useful for ownerdraw listboxes that do not make use of text strings.

LM_INSERTMULTITEMS Return Value - ICount

ICount ([LONG](#))

Number of items successfully inserted into the list.

If the number of items is not the same as *ulItemCount*, an error has occurred.

LM_INSERTMULTITEMS - Parameters

pListboxInfo ([PLBOXINFO](#))

Pointer to a structure containing list box information.

ppszText ([PSZ *](#))

Pointer to an array of pointers to text strings.

This parameter is a pointer to an array of pointers to zero-terminated strings. The array must contain at least *ulItemCount* items. (*ulItemCount* is a field in [LBOXINFO](#).)

If this parameter is set to NULL, a *ulItemCount* number of empty items are inserted into the list. This is useful for ownerdraw listboxes that do not make use of text strings.

lCount ([LONG](#))

Number of items successfully inserted into the list.

If the number of items is not the same as *ulItemCount*, an error has occurred.

LM_INSERTMULTITEMS - Syntax

This message inserts one or more items into a list box.

```
param1
    PLBOXINFO  pListboxInfo  /* Pointer to a structure containing list box information. */

param2
    PSZ *      papszText      /* Pointer to an array of pointers to text strings. */
```

LM_INSERTMULTITEMS - Remarks

LM_INSERTMULTITEMS inserts multiple items into a list box at one time, up to 32 768 items.

If either LIT_SORTASCENDING or LIT_SORTDESCENDING is specified in the *ItemIndex* field of [LBOXINFO](#), then the complete list is sorted after the items have been inserted. If items are being added using several LM_INSERTMULTITEMS messages, it is faster to specify LIT_END for all the insert messages except the last one, and then set one of the sort flags to sort the entire list after the last set of items have been inserted.

The sorting sequence is the same as that defined for [WinCompareStrings](#).

[WM_MEASUREITEM](#) (in List Boxes) is sent to the owner of an ownerdraw list box for every item inserted into the list box.

LM_INSERTMULTITEMS - Default Processing

The default message procedure sets *lCount* to zero.

LM_INSERTMULTITEMS - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

LM_QUERYITEMCOUNT

LM_QUERYITEMCOUNT Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

LM_QUERYITEMCOUNT Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

LM_QUERYITEMCOUNT Return Value - slItemCount

slItemCount (SHORT)
Item count.

LM_QUERYITEMCOUNT - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

slItemCount (SHORT)
Item count.

LM_QUERYITEMCOUNT - Syntax

This message returns a count of the number of items in the list box control.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */
```



```
param2
    ULONG  ulReserved  /* Reserved value, should be 0. */
```

LM_QUERYITEMCOUNT - Remarks

The list box control window procedure responds to this message by setting *sItemCount* to the number of items in the list.

LM_QUERYITEMCOUNT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sItemCount* to the default value of 0.

LM_QUERYITEMCOUNT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

LM_QUERYITEMHANDLE

LM_QUERYITEMHANDLE Field - sItemIndex

sItemIndex ([SHORT](#))
Item index.

LM_QUERYITEMHANDLE Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

LM_QUERYITEMHANDLE Return Value - ullItem

ullItem ([ULONG](#))
Item handle.

0	The indexed item does not exist.
Other	Item handle.

LM_QUERYITEMHANDLE - Parameters

sItemIndex ([SHORT](#))
Item index.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ullItem ([ULONG](#))
Item handle.

0	The indexed item does not exist.
Other	Item handle.

LM_QUERYITEMHANDLE - Syntax

This message returns the handle of the indexed item of the list box control.

```
param1
    SHORT sItemIndex /* Item index. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

LM_QUERYITEMHANDLE - Remarks

The meaning of the item handle is defined by the application. It may, for example, be a pointer to an application defined data structure.

Item handles are initialized to NULLHANDLE when an item is created. The list box control window procedure responds to this message by setting *ullItem* to the handle of the item whose index is specified by *sltemIndex*.

LM_QUERYITEMHANDLE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *ullItem* to the default value of NULLHANDLE.

The item handle is initialized to NULLHANDLE.

LM_QUERYITEMHANDLE - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

LM_QUERYITEMTEXT

LM_QUERYITEMTEXT Field - sltemIndex

sltemIndex ([SHORT](#))
Item index.

LM_QUERYITEMTEXT Field - smaxcount

smaxcount ([SHORT](#))
Maximum count.

- | | |
|-------|---|
| 0 | No text is copied. |
| Other | Copy the item text as a null-terminated string, but limit the number of characters copied, including the null termination character, to this value. |

LM_QUERYITEMTEXT Field - pszItemText

pszItemText ([PSZ](#))
Buffer into which the item text is to be copied.

This points to a string (character) buffer.

LM_QUERYITEMTEXT Return Value - sTextLength

sTextLength ([SHORT](#))
Length of item text.

The length of the text string, excluding the null termination character.

LM_QUERYITEMTEXT - Parameters

sItemIndex ([SHORT](#))
Item index.

smaxcount ([SHORT](#))
Maximum count.

0
Other

No text is copied.
Copy the item text as a null-terminated string, but limit the number of characters copied, including the null termination character, to this value.

pszItemText ([PSZ](#))
Buffer into which the item text is to be copied.

This points to a string (character) buffer.

sTextLength ([SHORT](#))
Length of item text.

The length of the text string, excluding the null termination character.

LM_QUERYITEMTEXT - Syntax

This message returns the text of the specified list box item.

```
param1
    SHORT sItemIndex    /* Item index. */
```

```
    SHORT    smaxcount    /* Maximum count. */

param2
    PSZ      pszItemText /* Buffer into which the item text is to be copied. */
```

LM_QUERYITEMTEXT - Remarks

The list box control window procedure responds to this message by copying up to *smaxcount* characters, as a null-terminated string, from the text of the item specified by *sItemIndex* into the buffer identified by *pszItemText*.

The length of the item text can be determined by using the [LM_QUERYITEMTEXTLENGTH](#) message.

LM_QUERYITEMTEXT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sTextLength* to the default value of 0.

LM_QUERYITEMTEXT - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

LM_QUERYITEMTEXTLENGTH

LM_QUERYITEMTEXTLENGTH Field - sItemIndex

sItemIndex ([SHORT](#))
Item index.

LM_QUERYITEMTEXTLENGTH Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

LM_QUERYITEMTEXTLENGTH Return Value - sTextLength

sTextLength (SHORT)
Length of item text.

The length of the text string, excluding the null termination character.

LIT_ERROR Error occurred. For example, the item specified by its index does not exist.

Other Length of item text.

LM_QUERYITEMTEXTLENGTH - Parameters

sItemIndex (SHORT)
Item index.

ulReserved (ULONG)
Reserved value, should be 0.

sTextLength (SHORT)
Length of item text.

The length of the text string, excluding the null termination character.

LIT_ERROR Error occurred. For example, the item specified by its index does not exist.

Other Length of item text.

LM_QUERYITEMTEXTLENGTH - Syntax

This message returns the length of the text of the specified list box item.

```
param1
    SHORT  sItemIndex    /* Item index. */

param2
    ULONG  ulReserved    /* Reserved value, should be 0. */
```

LM_QUERYITEMTEXTLENGTH - Remarks

The list box control window procedure responds to this message by setting *sTextLength* to the length in characters of the text of the item specified by *sItemIndex*.

LM_QUERYITEMTEXTLENGTH - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than set *sTextLength* to the default value of 0.

LM_QUERYITEMTEXTLENGTH - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

LM_QUERYSELECTION

LM_QUERYSELECTION Field - sltemStart

sltemStart ([SHORT](#))

Index of the start item.

If the list box allows multiple selected items, that is, if it has a style of LS_MULTIPLESEL, then this parameter indicates the index of the item from which the search for the next selected item is to begin. Therefore, to get all the selected items of the list, this message is sent repeatedly, each time setting this parameter to the index of the item returned by the previous usage of this message.

If this parameter is set to LIT_CURSOR the index of the item in the list box which currently has the cursor is returned.

If the list box only allows a single selection, this parameter is ignored.

LIT_CURSOR

Return the index of the item in the list box which currently has the cursor.

LIT_FIRST

Start the search at the first item.

Other

Start the search after the item specified by this index.

LM_QUERYSELECTION Field - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

LM_QUERYSELECTION Return Value - slItemSelected

slItemSelected (SHORT)

Index of the selected item.

LIT_NONE

No selected item.

For a single selection list box, this implies that there is no selected item in the list box. For a multiple selection list box, this implies that there is no selected item in the list box whose index is higher than the index specified by the *slItemStart* parameter.

Other

Index of selected item. For a single selection list box, this is the index of the only selected item in the list box. For a multiple selection list box, this is the index of the next selected item in the list box whose index is higher than the index specified by the *slItemStart* parameter.

If *slItemStart* is set to LIT_CURSOR, the index of the list-box item which currently has the cursor is returned.

LM_QUERYSELECTION - Parameters

slItemStart (SHORT)

Index of the start item.

If the list box allows multiple selected items, that is, if it has a style of LS_MULTIPLESEL, then this parameter indicates the index of the item from which the search for the next selected item is to begin. Therefore, to get all the selected items of the list, this message is sent repeatedly, each time setting this parameter to the index of the item returned by the previous usage of this message.

If this parameter is set to LIT_CURSOR the index of the item in the list box which currently has the cursor is returned.

If the list box only allows a single selection, this parameter is ignored.

LIT_CURSOR

Return the index of the item in the list box which currently has the cursor.

LIT_FIRST

Start the search at the first item.

Other

Start the search after the item specified by this index.

ulReserved (ULONG)

Reserved value, should be 0.

sItemSelected ([SHORT](#))

Index of the selected item.

LIT_NONE

No selected item.

For a single selection list box, this implies that there is no selected item in the list box. For a multiple selection list box, this implies that there is no selected item in the list box whose index is higher than the index specified by the *sItemStart* parameter.

Other

Index of selected item. For a single selection list box, this is the index of the only selected item in the list box. For a multiple selection list box, this is the index of the next selected item in the list box whose index is higher than the index specified by the *sItemStart* parameter.

If *sItemStart* is set to LIT_CURSOR, the index of the list-box item which currently has the cursor is returned.

LM_QUERYSELECTION - Syntax

This message is used to enumerate the selected item, or items, in a list box.

```
param1
    SHORT  sItemStart      /* Index of the start item. */

param2
    ULONG  ulReserved      /* Reserved value, should be 0. */
```

LM_QUERYSELECTION - Remarks

The list box control window procedure responds to this message by returning in *sItemSelected* the zero-based index of the selected item or next selected item after *sItemStart*, if any.

LM_QUERYSELECTION - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than set *sItemSelected* to the default value of 0.

LM_QUERYSELECTION - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

LM_QUERYTOPINDEX

LM_QUERYTOPINDEX Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

LM_QUERYTOPINDEX Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

LM_QUERYTOPINDEX Return Value - sltemTop

sltemTop ([SHORT](#))
Index of the item currently at the top of the list box:

LIT_NONE	No items in the list box
Other	Index of the item currently at the top of the list box.

LM_QUERYTOPINDEX - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

sltemTop ([SHORT](#))
Index of the item currently at the top of the list box:

LIT_NONE	No items in the list box
----------	--------------------------

Other

Index of the item currently at the top of the list box.

LM_QUERYTOPINDEX - Syntax

This message obtains the index of the item currently at the top of the list box.

```
param1
    ULONG    ulReserved /* Reserved value, should be 0. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */
```

LM_QUERYTOPINDEX - Remarks

The list box control window procedure responds to this message by returning in *sItemTop* the zero-based index of the item currently at the top of the list box.

LM_QUERYTOPINDEX - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sItemTop* to the default value of 0.

LM_QUERYTOPINDEX - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

LM_SEARCHSTRING

LM_SEARCHSTRING Field - uscmd

uscmd (USHORT)
Command.

Defines the criteria by which the string specified by the *pszSearchString* parameter is to be compared with the text of the items, to determine the index of the first matching item.

These values can be combined using the logical-OR operator:

LSS_CASESENSITIVE
Matching occurs if the item contains the characters specified by the *pszSearchString* parameter exactly.
This value is mandatory.

LSS_PREFIX
Matching occurs if the leading characters of the item contain the characters specified by the *pszSearchString* parameter.
If this value is specified, LSS_SUBSTRING must not be specified.

LSS_SUBSTRING
Matching occurs if the item contains a substring of the characters specified by the *pszSearchString* parameter.
If this value is specified, LSS_PREFIX must not be specified.

LM_SEARCHSTRING Field - sltemStart

sltemStart (SHORT)
Index of the start item.

LIT_FIRST
Start the search at the first item.

Other
Start the search after the item specified by this index.

LM_SEARCHSTRING Field - pszSearchString

pszSearchString (PSZ)
Search string.

This points to the string to search for.

LM_SEARCHSTRING Return Value - sltemMatched

sltemMatched (SHORT)
Index item whose text matches the string.

LIT_ERROR
Error occurred

LIT_NONE	No item found
Other	Index item whose text matches the string.

LM_SEARCHSTRING - Parameters

uscmd (USHORT)

Command.

Defines the criteria by which the string specified by the *pszSearchString* parameter is to be compared with the text of the items, to determine the index of the first matching item.

These values can be combined using the logical-OR operator:

LSS_CASESENSITIVE

Matching occurs if the item contains the characters specified by the *pszSearchString* parameter exactly.

This value is mandatory.

LSS_PREFIX

Matching occurs if the leading characters of the item contain the characters specified by the *pszSearchString* parameter.

If this value is specified, LSS_SUBSTRING must not be specified.

LSS_SUBSTRING

Matching occurs if the item contains a substring of the characters specified by the *pszSearchString* parameter.

If this value is specified, LSS_PREFIX must not be specified.

sItemStart (SHORT)

Index of the start item.

LIT_FIRST

Start the search at the first item.

Other

Start the search after the item specified by this index.

pszSearchString (PSZ)

Search string.

This points to the string to search for.

sItemMatched (SHORT)

Index item whose text matches the string.

LIT_ERROR

Error occurred

LIT_NONE

No item found

Other

Index item whose text matches the string.

LM_SEARCHSTRING - Syntax

This message returns the index of the list box item whose text matches the string.

```
param1
    USHORT   uscmd           /* Command. */
    SHORT    sItemStart      /* Index of the start item. */

param2
    PSZ      pszSearchString /* Search string. */
```

LM_SEARCHSTRING - Remarks

The list box control window procedure responds to this message by setting *sItemMatched* to the index of the next item whose text matches the string specified by *pszSearchString*.

All the items of the list are searched until a match is found, that is, the search wraps from the end to the start of the list.

LM_SEARCHSTRING - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sItemMatched* to the default value of 0.

LM_SEARCHSTRING - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

LM_SELECTITEM

LM_SELECTITEM Field - sltemIndex

sltemIndex ([SHORT](#))

Index of the item to be selected or deselected:

LIT_NONE

All items are to be deselected

Other

Index of the item to be selected or deselected.

LM_SELECTITEM Field - usselect

usselect (USHORT)

Select flag.

(Ignored if *sItemIndex* is set to LIT_NONE).

TRUE

The item is selected. If the control is a single selection list box (that is, it does not have the style of LS_MULTIPLESEL), any previously selected item is deselected.

FALSE

The item is deselected.

LM_SELECTITEM Return Value - rc

rc (BOOL)

Success indicator.

TRUE

Successful completion

FALSE

Error occurred. For example, when the item does not exist in the list box, or when an item that is not selected is deselected.

LM_SELECTITEM - Parameters

sItemIndex (SHORT)

Index of the item to be selected or deselected:

LIT_NONE

All items are to be deselected

Other

Index of the item to be selected or deselected.

usselect (USHORT)

Select flag.

(Ignored if *sItemIndex* is set to LIT_NONE).

TRUE

The item is selected. If the control is a single selection list box (that is, it does not have the style of LS_MULTIPLESEL), any previously selected item is deselected.

FALSE

The item is deselected.

rc (BOOL)

Success indicator.

TRUE	Successful completion
FALSE	Error occurred. For example, when the item does not exist in the list box, or when an item that is not selected is deselected.

LM_SELECTITEM - Syntax

This message is used to set the selection state of an item in a list box.

```
param1
    SHORT    sItemIndex /* Index of the item to be selected or deselected: */

param2
    USHORT   usselect /* Select flag. */
```

LM_SELECTITEM - Remarks

The list box control window procedure responds to this message by setting the selection state, as indicated by *usselect*, of the item whose index is specified in *sItemIndex*.

LM_SELECTITEM - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

LM_SELECTITEM - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

LM_SETITEMHANDLE

LM_SETITEMHANDLE Field - sltemIndex

sltemIndex ([SHORT](#))
Item index.

LM_SETITEMHANDLE Field - ulltemHandle

ulltemHandle ([ULONG](#))
Item handle.

LM_SETITEMHANDLE Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

LM_SETITEMHANDLE - Parameters

sltemIndex ([SHORT](#))
Item index.

ulltemHandle ([ULONG](#))
Item handle.

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

LM_SETITEMHANDLE - Syntax

This message sets the handle of the specified list box item.

```
param1
    SHORT  sItemIndex    /* Item index. */

param2
    ULONG  ulItemHandle  /* Item handle. */
```

LM_SETITEMHANDLE - Remarks

The meaning of the item handle is defined by the application. It may, for example, be a pointer to an application defined data structure.

Item handles are initialized to NULLHANDLE when an item is created.

The list box control window procedure responds to this message by setting the handle of the item whose index is specified by *sItemIndex* to the value specified by *ulItemHandle*.

LM_SETITEMHANDLE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

LM_SETITEMHANDLE - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

LM_SETITEMHEIGHT

LM_SETITEMHEIGHT Field - flNewHeight

flNewHeight ([ULONG](#))

Height of items in list box.

LM_SETITEMHEIGHT Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

LM_SETITEMHEIGHT Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful operation
FALSE	Error occurred.

LM_SETITEMHEIGHT - Parameters

flNewHeight (ULONG)
Height of items in list box.

ulReserved (ULONG)
Reserved value, should be 0.

rc (BOOL)
Success indicator.

TRUE	Successful operation
FALSE	Error occurred.

LM_SETITEMHEIGHT - Syntax

This message sets the height of the items in a list box.

```
param1
    ULONG flNewHeight /* Height of items in list box. */
param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

LM_SETITEMHEIGHT - Remarks

The list box control window procedure responds to this message by setting the height of the items in a list box to that specified by *lNewHeight*.

This message does *not* send a [WM_MEASUREITEM](#) message.

LM_SETITEMHEIGHT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *lC* to the default value of FALSE.

LM_SETITEMHEIGHT - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

LM_SETITEMTEXT

LM_SETITEMTEXT Field - *sltemIndex*

sltemIndex ([SHORT](#))

Item index.

LM_SETITEMTEXT Field - *pszItemText*

pszItemText ([PSZ](#))

Item text.

This points to a string containing the text to set the list-box item to.

LM_SETITEMTEXT Return Value - rc

rc (**BOOL**)

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

LM_SETITEMTEXT - Parameters

sItemIndex (**SHORT**)

Item index.

pszItemText (**PSZ**)

Item text.

This points to a string containing the text to set the list-box item to.

rc (**BOOL**)

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

LM_SETITEMTEXT - Syntax

This message sets the text into the specified list box item.

```
param1
    SHORT  sItemIndex    /* Item index. */

param2
    PSZ    pszItemText   /* Item text. */
```

LM_SETITEMTEXT - Remarks

The list box control window procedure responds to this message by copying the text identified by the *pszItemText* parameter into the item in

the list specified by the *s/item/index* parameter.

LM_SETITEMTEXT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

LM_SETITEMTEXT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

LM_SETITEMWIDTH

LM_SETITEMWIDTH Field - INewWidth

INewWidth ([ULONG](#))
Width of items in list box.

LM_SETITEMWIDTH Field - reserved

reserved ([ULONG](#))
Reserved value, should be 0.

LM_SETITEMWIDTH Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

LM_SETITEMWIDTH - Parameters

lNewWidth ([ULONG](#))
Width of items in list box.

reserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

LM_SETITEMWIDTH - Syntax

This message sets the width of the items in a list box.

```
param1
    ULONG lNewWidth /* Width of items in list box. */

param2
    ULONG reserved /* Reserved value, should be 0. */
```

LM_SETITEMWIDTH - Remarks

The list box control window procedure responds to this message by setting the width of the items in a list box to that specified by *lNewWidth*.

Note: Only list boxes with the LS_HORZSCROLL style set will respond to this message.

This message does *not* send a [WM_MEASUREITEM](#) message.

LM_SETITEMWIDTH - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the

default value of FALSE.

LM_SETITEMWIDTH - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

LM_SETTOPINDEX

LM_SETTOPINDEX Field - sltemIndex

sltemIndex ([SHORT](#))
Index of the item to be made top.

LM_SETTOPINDEX Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

LM_SETTOPINDEX Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

LM_SETTOPINDEX - Parameters

sItemIndex ([SHORT](#))
Index of the item to be made top.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

LM_SETTOPINDEX - Syntax

This message is used to scroll a particular item to the top of the list box.

```
param1
    SHORT sItemIndex /* Index of the item to be made top. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

LM_SETTOPINDEX - Remarks

The list box control window procedure responds to this message by scrolling the item whose index is identified by *sItemIndex* to the top of the list box.

LM_SETTOPINDEX - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

LM_SETTOPINDEX - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_CHAR (in List Boxes)

WM_CHAR (in List Boxes) - Syntax

For the cause of this message, see [WM_CHAR](#).

For a description of the parameters, see [WM_CHAR](#).

WM_CHAR (in List Boxes) - Remarks

The list box control window procedure responds to this message by sending it to its owner if it has not processed the key stroke. This is the most common means by which the input focus is switched around the various controls in a dialog box.

The key strokes processed by a list box control are:

Down Arrow	Moves the selection down one item, scrolling the list box by one item, if necessary, to make the next item visible. When the selection reaches the bottom, the Down Arrow has no effect.
Up Arrow	Moves the selection up one item, scrolling the list box by one item, if necessary, to make the previous item visible. When the selection reaches the top, the Up Arrow has no effect.
Page Down	Moves the selection down one page, scrolling the list box by the number of items visible in the list box. For example, if the list box displays seven items and item 1 is selected and positioned at the top of the list box, pressing the Page Down key causes item 8 to be selected and displayed at the top of the list box. Pressing Page Down when the last item is selected has no effect.
Page Up	Moves the selection up one page, scrolling the list box by the number of items visible in the list box. For example, if the list box displays seven items and item 8 is selected and positioned at the top of the list box, pressing the Page Up key causes item 1 to be selected and displayed at the top of the list box. Pressing the Page Up key when the first item is selected has no effect.

WM_CHAR (in List Boxes) - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE

WM_CHAR (in List Boxes) - Related Messages

Related Messages

- [WM_CHAR](#)

WM_CHAR (in List Boxes) - Topics

Select an item:

[Syntax](#)

[Remarks](#)

[Default Processing](#)

[Related Messages](#)

[Glossary](#)

WM_QUERYCONVERTPOS (in List Boxes)

WM_QUERYCONVERTPOS (in List Boxes) - Syntax

For the cause of this message, see [WM_QUERYCONVERTPOS](#).

For a description of the parameters, see [WM_QUERYCONVERTPOS](#).

WM_QUERYCONVERTPOS (in List Boxes) - Remarks

The list box control window procedure returns QCP_NOCONVERT.

WM_QUERYCONVERTPOS (in List Boxes) - Default Processing

For the default window procedure processing of this message see [WM_QUERYCONVERTPOS](#).

WM_QUERYCONVERTPOS (in List Boxes) - Related Messages

Related Messages

- [WM_QUERYCONVERTPOS](#)

WM_QUERYCONVERTPOS (in List Boxes) - Topics

Select an item:

[Syntax](#)

[Remarks](#)

[Default Processing](#)

[Related Messages](#)

[Glossary](#)

WM_QUERYWINDOWPARAMS (in List Boxes)

WM_QUERYWINDOWPARAMS (in List Boxes) - Syntax

Occurs when an application queries the list box control window parameters.

For a description of the parameters, see [WM_QUERYWINDOWPARAMS](#).

WM_QUERYWINDOWPARAMS (in List Boxes) - Remarks

The list box control window procedure responds to this message by passing it to the default window procedure.

WM_QUERYWINDOWPARAMS (in List Boxes) - Default Processing

The default window procedure sets the *cchText*, *cbPresParams*, and *cbCtlData* parameters of the [WNDPARAMS](#) data structure, identified by *pwndparams*, to 0 and sets *rc* to FALSE.

WM_QUERYWINDOWPARAMS (in List Boxes) - Related Messages

Related Messages

- [WM_QUERYWINDOWPARAMS](#)
-

WM_QUERYWINDOWPARAMS (in List Boxes) - Topics

Select an item:

[Syntax](#)

[Remarks](#)

[Default Processing](#)

[Related Messages](#)

[Glossary](#)

WM_SETWINDOWPARAMS (in List Boxes)

WM_SETWINDOWPARAMS (in List Boxes) - Syntax

This message occurs when an application sets or changes the list box control window parameters.

For a description of the parameters, see [WM_SETWINDOWPARAMS](#).

WM_SETWINDOWPARAMS (in List Boxes) - Remarks

The list box control window procedure responds to this message by passing it to the default window procedure.

WM_SETWINDOWPARAMS (in List Boxes) - Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

WM_SETWINDOWPARAMS (in List Boxes) - Related Messages

Related Messages

- [WM_SETWINDOWPARAMS](#)

WM_SETWINDOWPARAMS (in List Boxes) - Topics

Select an item:

[Syntax](#)

[Remarks](#)

[Default Processing](#)

[Related Messages](#)

[Glossary](#)

Menu Control Window Processing

This system-provided window procedure processes the actions on a menu control (WC_MENU).

Purpose

A menu control is a child or pull-down window that contains a list of selection items. These items can be represented by text strings, separators, bit maps or menu buttons. Menu templates can be loaded as resources and the menu can be created automatically when the parent window is created. The application can build the menu dynamically by sending [MM_INSERTITEM](#) messages. An application can change a menu by sending messages to it.

Menus enable the operator to select one of the items in the list, using the pointing device or the keyboard. When a selection is made, the menu parent is notified by posting a [WM_COMMAND](#), [WM_SYSCOMMAND](#), or [WM_HELP](#) message and a unique identifier representing the operator's selection.

Menus automatically resize themselves when items are added and removed. Menus are automatically destroyed when their owner is destroyed.

Typically, an application has an action bar menu and several submenus. The action bar is normally visible, and is a child window in the parent window frame. The submenus are normally hidden and become visible when selections are made on the action bar.

Menu Control Styles

These menu control styles are available:

MS_ACTIONBAR

The items in the list are displayed side-by-side. This style is used to implement a top level menu. Menus that do not have this style are displayed in one or more columns and are submenus associated with an action bar.

All menu controls have styles [CS_SYNCPAINT](#) and [CS_PARENTCLIP](#).

MS_CONDITIONALCASCADE

This style is used to specify that the items in this list are a conditional cascade menu. Conditional cascade menus act like normal cascade menus with the exception that the cascade does not automatically open when the user selects it. To open the conditional cascade menu, the mini-pushbutton on the menu item must be selected. If the menu is selected without opening the cascade, the default item in the cascade is selected. The default action on the cascade is identified by a check mark.

MS_TITLEBUTTON

Used to identify menus that can be used as buttons in the title bar. Can only be used with [MS_ACTIONBAR](#).

This style causes the menu to be drawn using the CUA colors specified for the title bar rather than the action bar.

MS_VERTICALFLIP

Normally, pull-down menus (the default, without the [MS_VERTICALFLIP](#) style) are displayed below their associated action bar item. If there is not room on the screen to display the entire pull-down in this manner, and if there is room to display the pull-down above the action bar, it is displayed above the action bar. Pull-down menus with the [MS_VERTICALFLIP](#) style are flipped vertically. That is, they are displayed above the menu if possible, otherwise below it. The vertical flip style must be set explicitly by the application when the window is minimized, and must be reset when it is restored.

If an application action bar contains this style, the style is applied to all pull-down menus belonging to the action bar (the style does not directly affect the display of the action bar). This provides a convenient means for the application to flip the appearance of all pull-down menus.

Menu Item Styles

These menu item styles are available:

MIS_SUBMENU

The item is a submenu. When the user selects this type of item, a submenu is displayed from which the user must make further selection. Items that are not submenu items are command items.

MIS_SEPARATOR

The display object is a horizontal dividing line. This type of item can only be used in pull-down menus. This type of item cannot be enabled, checked, disabled, highlighted, or selected by the user. The functional object is NULL when this style is specified.

MIS_BITMAP

The display object is a bit map.

MIS_TEXT

The display object is a text string.

MIS_BUTTONSEPARATOR

The item is a menu button. Any menu can have zero, one, or two items of this type. These are the last items in a menu and are automatically displayed after a separator bar. The user cannot move the cursor to these items, but can select them with the pointing device or with the appropriate key.

MIS_BREAK

The item begins a new row or column.

MIS_BREAKSEPARATOR

Same as MIS_BREAK, except that it draws a separator between rows or columns of a pull-down menu. This style can only be used within a submenu.

MIS_SYSCOMMAND

If this item is selected, the menu notifies the owner by posting a [WM_SYSCOMMAND](#) message rather than a [WM_COMMAND](#) message.

MIS_OWNERDRAW

Items with this style are drawn by the owner. [WM_DRAWITEM](#) and [WM_MEASUREITEM](#) notification messages are sent to the owner to draw the item or determine its size.

MIS_HELP

If the item is selected, the menu notifies the owner by posting a [WM_HELP](#) message rather than a [WM_COMMAND](#) message.

MIS_STATIC

This type of item exists for information purposes only. It cannot be selected with the pointing device or keyboard.

Menu Item Attributes

Applications can get and set the state of these attributes by sending [MM_QUERYITEMATTR](#) and [MM_SETITEMATTR](#) messages.

These menu item attributes are available:

MIA_HILITED

The state of this attribute is TRUE, if and only if, the item is selected.

MIA_CHECKED

If this attribute is TRUE, a check mark appears next to the item (submenu only).

MIA_DISABLED

This attribute is TRUE if the item is disabled and cannot be selected. The item is drawn in a disabled state.

MIA_FRAMED

If this attribute is TRUE, a frame is drawn around the item (top-level menu only).

MIA_NODISMISS

If the item is selected, the submenu remains down. A menu with this attribute is not hidden until the application or user explicitly does so, for example by selecting either another menu on the action bar or by pressing the escape key.

Default Colors

The following system colors are used when the system draws menu controls:

SYSCLR_WINDOWFRAME
SYSCLR_BUTTONDARK
SYSCLR_BUTTONLIGHT
SYSCLR_SHADOW
SYSCLR_TITLEBOTTOM
SYSCLR_DIALOGBACKGROUND

Some of these defaults can be replaced by using the following presentation parameters in the application resource script file or source code:

PP_FOREGROUNDCOLOR
PP_HILITEFOREGROUNDCOLOR
PP_BORDERCOLOR
PP_DISABLEDFOREGROUNDCOLOR

Menu Control Notification Messages

These messages are initiated by the menu control window procedure to notify its owner of significant events.

WM_COMMAND (in Menu Controls)

WM_COMMAND (in Menu Controls) - Syntax

For the cause of this message, see [WM_COMMAND](#).

For a description of the parameters, see [WM_COMMAND](#).

The menu control window procedure sets *uscmd* to the menu-item identity.

WM_COMMAND (in Menu Controls) - Remarks

The menu control window procedure generates this message if the [WM_MENUSELECT \(in Menu Controls\)](#) message returns a *rc* of TRUE.

when an item is selected that does not have the style of MIS_SYSCOMMAND or MIS_HELP. The menu control window procedure posts the message to the queue of the window owner.

WM_COMMAND (in Menu Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *uiReserved* to 0.

WM_COMMAND (in Menu Controls) - Related Messages

Related Messages

- [WM_COMMAND](#)
-

WM_COMMAND (in Menu Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_DRAWITEM (in Menu Controls)

WM_DRAWITEM (in Menu Controls) Field - idMenu

idMenu ([USHORT](#))
Window identifier.

The window identity of the menu control sending this notification message.

WM_DRAWITEM (in Menu Controls) Field - pOwnerItem

pOwnerItem ([POWNERITEM](#))

Owner-item structure.

This points to an owner-item structure; see [OWNERITEM](#).

WM_DRAWITEM (in Menu Controls) Return Value - rc

rc ([BOOL](#))

Item-drawn indicator.

TRUE

The owner draws the item, and so the menu control does not draw it.

FALSE

If the item contains text and the owner does not draw the item, the owner returns this value and the menu control draws the item.

WM_DRAWITEM (in Menu Controls) - Parameters

idMenu ([USHORT](#))

Window identifier.

The window identity of the menu control sending this notification message.

pOwnerItem ([POWNERITEM](#))

Owner-item structure.

This points to an owner-item structure; see [OWNERITEM](#).

rc ([BOOL](#))

Item-drawn indicator.

TRUE

The owner draws the item, and so the menu control does not draw it.

FALSE

If the item contains text and the owner does not draw the item, the owner returns this value and the menu control draws the item.

WM_DRAWITEM (in Menu Controls) - Syntax

This notification is sent to the owner of a menu control each time an item is to be drawn.

```
param1
    USHORT      idMenu      /* Window identifier. */

param2
    POWNERITEM  pOwnerItem  /* Owner-item structure. */
```

WM_DRAWITEM (in Menu Controls) - Remarks

The menu control window procedure only draws items that are represented by text strings and emphasizes selected items by inverting them.

If an application uses menu controls containing items that are not represented by text strings, or requires that the emphasized state of an item is to be drawn in a special manner, then the menu control must specify the style `MIS_OWNERDRAW` and those items must be drawn by the owner.

The menu control window procedure generates this message and sends it to its owner, informing the owner that an item is to be drawn, offering the owner the opportunity to draw that item, and to indicate that either the item has been drawn, or that the menu control is to draw it.

WM_DRAWITEM (in Menu Controls) - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set `rc` to the default value of `FALSE`.

WM_DRAWITEM (in Menu Controls) - Related Messages

Related Messages

- [WM_DRAWITEM](#)

WM_DRAWITEM (in Menu Controls) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_HELP (in Menu Controls)

WM_HELP (in Menu Controls) - Syntax

For the cause of this message, see [WM_HELP](#).

For a description of the parameters, see [WM_HELP](#).

The menu control window procedure sets *uscmd* to the menu-item identity.

WM_HELP (in Menu Controls) - Remarks

This message is identical to a [WM_COMMAND](#) message, but implies that the application should respond to this message by displaying help information.

The menu control window procedure generates this message and posts it to the queue of its owner when an item is selected that has the style of MIS_HELP, but only if [WM_MENUSELECT \(in Menu Controls\)](#) returns a *rc* of TRUE.

WM_HELP (in Menu Controls) - Default Processing

The default window procedure sends this message to the parent window, if it exists and is not the desktop. Otherwise, it sets *uReserved* to 0.

WM_HELP (in Menu Controls) - Related Messages

Related Messages

- [WM_HELP](#)
-

WM_HELP (in Menu Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_INITMENU (in Menu Controls)

WM_INITMENU (in Menu Controls) - Syntax

For the cause of this message, see [WM_INITMENU](#).

For a description of the parameters, see [WM_INITMENU](#).

WM_INITMENU (in Menu Controls) - Remarks

This message offers the owner the opportunity to perform some initialization on the menu items before they are presented.

The menu control window procedure generates this message and sends it to its owner, informing the owner of the event.

WM_INITMENU (in Menu Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *u/Reserved* to 0.

WM_INITMENU (in Menu Controls) - Related Messages

Related Messages

- [WM_INITMENU](#)
-

WM_INITMENU (in Menu Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_MEASUREITEM (in Menu Controls)

WM_MEASUREITEM (in Menu Controls) Field - sMenu

sMenu ([SHORT](#))
Menu identifier.

WM_MEASUREITEM (in Menu Controls) Field - pOwnerItem

pOwnerItem ([POWNERITEM](#))
Owner-item structure.

This points to an [OWNERITEM](#) structure.

WM_MEASUREITEM (in Menu Controls) Return Value - sHeight

sHeight ([SHORT](#))
Height of item.

WM_MEASUREITEM (in Menu Controls) - Parameters

sMenu ([SHORT](#))
Menu identifier.

pOwnerItem ([POWNERITEM](#))
Owner-item structure.

This points to an [OWNERITEM](#) structure.

sHeight ([SHORT](#))
Height of item.

WM_MEASUREITEM (in Menu Controls) - Syntax

This notification is sent to the owner of a menu control to establish the height for an item in that control.

```
param1  
    SHORT          sMenu          /* Menu identifier. */  
  
param2  
    POWNERITEM pOwnerItem /* Owner-item structure. */
```

WM_MEASUREITEM (in Menu Controls) - Remarks

This message is only sent at the time the menu control is created. When the owner receives this message, it must calculate and return the height of an item to the control.

All items in a menu must have the same height, and that must be greater than or equal to the height of the current font.

In particular, this notification is sent to the owner of a menu that has a style of `MIS_OWNERDRAW`, to offer the owner an opportunity to establish the height of an item that accommodates any special requirements for the drawing of items in that menu. When the owner receives this message, it must calculate and return the height and width of an item to the control. Typically, the client window procedure processes `WM_MEASUREITEM` by filling in the *yTop* and *Right* fields of the `RECT` structure specified by the *rcItem* field of the `OWNERITEM` structure; this specifies the size of the rectangle needed to enclose the item when it is drawn.

WM_MEASUREITEM (in Menu Controls) - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *Height* to the default value of 0.

WM_MEASUREITEM (in Menu Controls) - Related Messages

Related Messages

- [WM_MEASUREITEM](#)

WM_MEASUREITEM (in Menu Controls) - Examples

The following code fragment responds to a `WM_MEASUREITEM` message.

```
case WM_MEASUREITEM:
    ((POWNERITEM) mp2)->rcItem.xRight = 26;
    ((POWNERITEM) mp2)->rcItem.yTop = 10;
    return 0;
```

WM_MEASUREITEM (in Menu Controls) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Examples](#)
[Glossary](#)

WM_MENUEND (in Menu Controls)

WM_MENUEND (in Menu Controls) - Syntax

For the cause of this message, see [WM_MENUEND](#).

For a description of the parameters, see [WM_MENUEND](#).

WM_MENUEND (in Menu Controls) - Remarks

The menu control window procedure generates this message and sends it to its owner, informing the owner of this event.

WM_MENUEND (in Menu Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *uIfReserved* to 0.

WM_MENUEND (in Menu Controls) - Related Messages

Related Messages

- [WM_MENUEND](#)
-

WM_MENUEND (in Menu Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_MENUSELECT (in Menu Controls)

WM_MENUSELECT (in Menu Controls) - Syntax

For the cause of this message, see [WM_MENUSELECT](#).

For a description of the parameters, see [WM_MENUSELECT](#).

WM_MENUSELECT (in Menu Controls) - Remarks

The menu control window procedure generates this message and sends it to its owner, informing the owner of this event.

When the message is returned from its owner, menu control acts on *rc* as appropriate.

It must not be posted to the menu control.

WM_MENUSELECT (in Menu Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to TRUE.

WM_MENUSELECT (in Menu Controls) - Related Messages

Related Messages

- [WM_MENUSELECT](#)
-

WM_MENUSELECT (in Menu Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_NEXTMENU (in Menu Controls)

WM_NEXTMENU (in Menu Controls) - Syntax

For the cause of this message, see [WM_NEXTMENU](#).

For a description of the parameters, see [WM_NEXTMENU](#).

WM_NEXTMENU (in Menu Controls) - Remarks

The menu control generates this message and sends it to its owner, informing the owner of this event.

WM_NEXTMENU (in Menu Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *hwndNewMenu* to NULLHANDLE.

WM_NEXTMENU (in Menu Controls) - Related Messages

Related Messages

- [WM_NEXTMENU](#)
-

WM_NEXTMENU (in Menu Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

Menu Control Window Messages

This section describes the menu control window procedure actions on receiving the following messages.

MM_DELETEITEM

MM_DELETEITEM Field - usitem

usitem (**USHORT**)
Item identifier.

MM_DELETEITEM Field - usincludesubmenus

usincludesubmenus (**USHORT**)
Include submenus indicator.

- TRUE If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and delete it.
- FALSE If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.
-

MM_DELETEITEM Field - ulReserved

ulReserved (**ULONG**)
Reserved value, should be 0.

MM_DELETEITEM Return Value - slItemsLeft

slItemsLeft (**SHORT**)
Number remaining.

The number of items in the menu after the item is deleted.

MM_DELETEITEM - Parameters

usitem (**USHORT**)
Item identifier.

usincludesubmenus (**USHORT**)
Include submenus indicator.

TRUE

If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and delete it.

FALSE

If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

ulReserved ([ULONG](#))

Reserved value, should be 0.

sItemsLeft ([SHORT](#))

Number remaining.

The number of items in the menu after the item is deleted.

MM_DELETEITEM - Syntax

This message deletes a menu item.

```
param1
    USHORT  usitem          /* Item identifier. */
    USHORT  usincludesubmenus /* Include submenus indicator. */

param2
    ULONG   ulReserved      /* Reserved value, should be 0. */
```

MM_DELETEITEM - Remarks

The menu control window procedure responds to this message by deleting the identified item from the menu or its submenus.

Note: It must be sent, not posted, to the menu control.

MM_DELETEITEM - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sItemsLeft* to the default value of 0.

MM_DELETEITEM - Topics

Select an item:

[Syntax](#)

[Parameters](#)

MM_ENDMENU MODE

MM_ENDMENU MODE Field - usdismiss

usdismiss ([USHORT](#))
Dismiss menu indicator.

TRUE

Dismiss the submenu or subdialog window

FALSE

Do not dismiss the submenu or subdialog window.

MM_ENDMENU MODE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MM_ENDMENU MODE Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MM_ENDMENU MODE - Parameters

usdismiss ([USHORT](#))
Dismiss menu indicator.

TRUE

Dismiss the submenu or subdialog window

FALSE

Do not dismiss the submenu or subdialog window.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

MM_ENDMENU MODE - Syntax

This message is sent to a menu control to terminate menu selection.

```
param1
    USHORT  usdismiss /* Dismiss menu indicator. */

param2
    ULONG   ulReserved /* Reserved value, should be 0. */
```

MM_ENDMENU MODE - Remarks

The menu control window procedure responds to this message by terminating menu selection.

Note: It must be sent, not posted, to the menu control.

MM_ENDMENU MODE - Default Processing

The default window procedure does not expect to receive this message and, therefore, takes no action on it, other than to set *ulReserved* to the default value of 0.

MM_ENDMENU MODE - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MM_INSERTITEM

MM_INSERTITEM Field - pmenuitem

pmenuitem ([PMENUITEM](#))

Menu-item data structure.

This points to a [MENUITEM](#) structure.

MM_INSERTITEM Field - pszItemText

pszItemText ([PSZ](#))

Item text.

This points to a string containing the text to be inserted.

MM_INSERTITEM Return Value - sIndexInserted

sIndexInserted ([SHORT](#))

Index of inserted item.

MIT_MEMERROR

The menu control cannot allocate space to insert the menu item in the menu.

MIT_ERROR

An error other than MIT_MEMERROR occurred.

Other

The zero-based index of the offset of the item within the menu.

MM_INSERTITEM - Parameters

pmenuitem ([PMENUITEM](#))

Menu-item data structure.

This points to a [MENUITEM](#) structure.

pszItemText ([PSZ](#))

Item text.

This points to a string containing the text to be inserted.

sIndexInserted ([SHORT](#))

Index of inserted item.

MIT_MEMERROR

The menu control cannot allocate space to insert the menu item in the menu.

MIT_ERROR

An error other than MIT_MEMERROR occurred.

Other

The zero-based index of the offset of the item within the menu.

MM_INSERTITEM - Syntax

This message inserts a menu item into a menu.

```
param1
    PMENUITEM  pmenuitem      /* Menu-item data structure. */

param2
    PSZ        pszItemText    /* Item text. */
```

MM_INSERTITEM - Remarks

The menu control window procedure responds to this message by inserting the identified item into the menu at the position indicated by the specified [MENUITEM](#) data structure (contained within the menu-item structure). If the position is MIT_END, the item is added to the end of the menu. If the style of the item includes MIS_TEXT, the text of the item is specified by *pszItemText*

The menu control window procedure sets *sIndexInserted* to the zero-based index of the position of the item within the menu.

Note: It must be sent, not posted, to the menu control.

MM_INSERTITEM - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sIndexInserted* to the default value of 0.

MM_INSERTITEM - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

MM_ISITEMVALID

MM_ISITEMVALID Field - usitem

usitem ([USHORT](#))
Item identifier.

MM_ISITEMVALID Field - usincludesubmenus

usincludesubmenus ([USHORT](#))
Include submenu indicator.

TRUE	If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier.
FALSE	If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

MM_ISITEMVALID Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MM_ISITEMVALID Return Value - rc

rc ([BOOL](#))
Selectable indication.

- A menu item can be selected and entered under these conditions:
- The item is enabled and, if it is a submenu item, the item in the action bar associated with the submenu is enabled. If the action bar item is not enabled, the user cannot display the submenu.

- The item is enabled, and the submenu is displayed and being tracked with the pointing device or keyboard. It is unlikely, but possible, that the associated action bar is disabled in this instance.

TRUE

The user can select and enter the specified item.

FALSE

The user cannot select and enter the specified item.

MM_ISITEMVALID - Parameters

usitem ([USHORT](#))

Item identifier.

usincludesubmenus ([USHORT](#))

Include submenus indicator.

TRUE

If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier.

FALSE

If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

ulReserved ([ULONG](#))

Reserved value, should be 0.

rc ([BOOL](#))

Selectable indication.

A menu item can be selected and entered under these conditions:

- The item is enabled and, if it is a submenu item, the item in the action bar associated with the submenu is enabled. If the action bar item is not enabled, the user cannot display the submenu.
- The item is enabled, and the submenu is displayed and being tracked with the pointing device or keyboard. It is unlikely, but possible, that the associated action bar is disabled in this instance.

TRUE

The user can select and enter the specified item.

FALSE

The user cannot select and enter the specified item.

MM_ISITEMVALID - Syntax

This message returns the selectable status of a specified menu item.

```
param1
    USHORT  usitem           /* Item identifier. */
    USHORT  usincludesubmenus /* Include submenus indicator. */

param2
    ULONG   ulReserved       /* Reserved value, should be 0. */
```

MM_ISITEMVALID - Remarks

The menu control window procedure responds to this message by setting the return value depending on the selectable status of the specified item.

MM_ISITEMVALID - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

MM_ISITEMVALID - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

MM_ITEMIDFROMPOSITION

MM_ITEMIDFROMPOSITION Field - sltemIndex

sltemIndex ([SHORT](#))

Item index.

MM_ITEMIDFROMPOSITION Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

MM_ITEMIDFROMPOSITION Return Value - sIdentity

sIdentity (SHORT)	Item identity.
MIT_ERROR	Error occurred; for example, because <i>sItemIndex</i> is not valid.
Other	Item identity.

MM_ITEMIDFROMPOSITION - Parameters

sItemIndex (SHORT)	Item index.
ulReserved (ULONG)	Reserved value, should be 0.
sIdentity (SHORT)	Item identity.
MIT_ERROR	Error occurred; for example, because <i>sItemIndex</i> is not valid.
Other	Item identity.

MM_ITEMIDFROMPOSITION - Syntax

This message returns the identity of a menu item of a specified index.

```
param1
    SHORT  sItemIndex /* Item index. */

param2
    ULONG  ulReserved /* Reserved value, should be 0. */
```

MM_ITEMIDFROMPOSITION - Remarks

The menu control window procedure responds to this message by setting *sIdentity* to the identity of the item whose position is identified by the index specified in *sItemIndex*.

Note: It must be sent, not posted, to the menu control.

MM_ITEMIDFROMPOSITION - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *s/identity* to the default value of 0.

MM_ITEMIDFROMPOSITION - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MM_ITEMPOSITIONFROMID

MM_ITEMPOSITIONFROMID Field - usitem

usitem ([USHORT](#))
Item identifier.

MM_ITEMPOSITIONFROMID Field - usincludesubmenus

usincludesubmenus ([USHORT](#))
Include submenus indicator.

- | | |
|-------|---|
| TRUE | If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier. |
| FALSE | If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier. |

MM_ITEMPOSITIONFROMID Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MM_ITEMPOSITIONFROMID Return Value - sIndex

sIndex (SHORT)
Item index.

MIT_NONE	Item does not exist
Other	Item index.

MM_ITEMPOSITIONFROMID - Parameters

usitem (USHORT)
Item identifier.

usincludesubmenus (USHORT)
Include submenus indicator.

TRUE	If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier.
FALSE	If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

ulReserved (ULONG)
Reserved value, should be 0.

sIndex (SHORT)
Item index.

MIT_NONE	Item does not exist
Other	Item index.

MM_ITEMPOSITIONFROMID - Syntax

This message returns the index of a menu item of a particular identity.

```
param1
    USHORT    usitem           /* Item identifier. */
```

```
        USHORT  usIncludesSubmenus  /* Include submenus indicator. */
param2
        ULONG   ulReserved          /* Reserved value, should be 0. */
```

MM_ITEMPOSITIONFROMID - Remarks

The menu control window procedure responds to this message by setting *sIndex* to the zero-based index of the item identified by *sIndex*.

Note: It must be sent, not posted, to the menu control.

Note: If MM_ITEMPOSITIONFROMID is sent to a window that has been destroyed, 0 is returned. The existence of an invalid window handle can be verified through [WinIsWindow](#).

MM_ITEMPOSITIONFROMID - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sIndex* to the default value of MIT_NONE.

MM_ITEMPOSITIONFROMID - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MM_QUERYDEFAULTITEMID

MM_QUERYDEFAULTITEMID Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, must be 0.

MM_QUERYDEFAULTITEMID Field - ulReserved

ulReserved (ULONG)
Reserved value, must be 0.

MM_QUERYDEFAULTITEMID Return Value - ulDefltemID

ulDefltemID (ULONG)
Menu id of the default menu item.

MM_QUERYDEFAULTITEMID - Parameters

ulReserved (ULONG)
Reserved value, must be 0.

ulReserved (ULONG)
Reserved value, must be 0.

ulDefltemID (ULONG)
Menu id of the default menu item.

MM_QUERYDEFAULTITEMID - Syntax

This message returns the default item id for a conditional cascade menu. For any other type of menu or submenu, this message returns zero.

```
param1
    ULONG ulReserved /* Reserved value, must be 0. */

param2
    ULONG ulReserved /* Reserved value, must be 0. */
```

MM_QUERYDEFAULTITEMID - Default Processing

The default window procedure takes no action other than to return 0.

MM_QUERYDEFAULTITEMID - Related Messages

Related Messages

- [WM_DRAWITEM](#) (in Frame Controls)
- [WM_DRAWITEM](#) (in List Boxes)
- [WM_DRAWITEM](#) (in Menu Controls)

MM_QUERYDEFAULTITEMID - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

MM_QUERYITEM

MM_QUERYITEM Field - usitem

usitem ([USHORT](#))
Item identifier.

MM_QUERYITEM Field - usincludesubmenus

usincludesubmenus ([USHORT](#))
Include submenus flag.

TRUE

If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and copy its definition.

FALSE

If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

MM_QUERYITEM Field - pmenuitem

pmenuitem (PMENUITEM)
Menu-item data structure.

This points to a MENUITEM structure.

MM_QUERYITEM Return Value - rc

rc (BOOL)
Success indicator.

TRUE Successful completion
FALSE Error occurred.

MM_QUERYITEM - Parameters

usitem (USHORT)
Item identifier.

usincludesubmenus (USHORT)
Include submenus flag.

TRUE If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and copy its definition.

FALSE If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

pmenuitem (PMENUITEM)
Menu-item data structure.

This points to a MENUITEM structure.

rc (BOOL)
Success indicator.

TRUE Successful completion
FALSE Error occurred.

MM_QUERYITEM - Syntax

This message returns the definition of the specified menu item.

```
param1
    USHORT    usitem          /* Item identifier. */
    USHORT    usincludesubmenus /* Include submenus flag. */

param2
    PMENUITEM pmenuitem      /* Menu-item data structure. */
```

MM_QUERYITEM - Remarks

The menu control window procedure responds to this message by copying the item definition specified by *usitem*, from the menu, to the structure specified by *pmenuitem*.

Note: This message does not retrieve the text for items with a style of MIS_TEXT. The item text is obtained by use of the [MM_QUERYITEMTEXT](#) message.

MM_QUERYITEM - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

MM_QUERYITEM - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MM_QUERYITEMATTR

MM_QUERYITEMATTR Field - usitem

usitem (USHORT)
Item identity.

MM_QUERYITEMATTR Field - usIncludeSubmenus

usIncludeSubmenus (USHORT)
Include submenus indicator.

- TRUE
- If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and return its state.
- FALSE
- If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

MM_QUERYITEMATTR Field - usattributemask

usattributemask (USHORT)
Attribute mask.

MM_QUERYITEMATTR Return Value - usState

usState (USHORT)
State.

MM_QUERYITEMATTR - Parameters

usitem (USHORT)
Item identity.

usIncludeSubmenus (USHORT)
Include submenus indicator.

- TRUE
- If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and return its state.
- FALSE
- If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

usattributemask (USHORT)
Attribute mask.

usState ([USHORT](#))
State.

MM_QUERYITEMATTR - Syntax

This message returns the attributes of a menu item.

```
param1
    USHORT  usitem          /* Item identity. */
    USHORT  usIncludeSubmenus /* Include submenus indicator. */

param2
    USHORT  usattributemask /* Attribute mask. */
```

MM_QUERYITEMATTR - Remarks

The menu control responds to this message by returning the state of the specified attributes of the identified menu item.

MM_QUERYITEMATTR - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *usState* to the default value of 0.

MM_QUERYITEMATTR - Examples

This example sends an MM_QUERYITEMATTR message to find the state of the 'idCase' menu item. It then toggles the state of the item and sends an MM_SETITEMATTR message to set the new state.

```
sState = (SHORT) WinSendMsg(hwndMenu, MM_QUERYITEMATTR,
    MPFROM2SHORT(idCase, TRUE), MPFROMSHORT(MIA_CHECKED));
sState ^= MIA_CHECKED;
WinSendMsg(hwndMenu, MM_SETITEMATTR, MPFROM2SHORT(idCase, TRUE),
    MPFROM2SHORT(MIA_CHECKED, sState));
```

MM_QUERYITEMATTR - Topics

Select an item:
[Syntax](#)

[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Examples](#)
[Glossary](#)

MM_QUERYITEMCOUNT

MM_QUERYITEMCOUNT Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MM_QUERYITEMCOUNT Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MM_QUERYITEMCOUNT Return Value - sresult

sresult ([SHORT](#))
Item count.

MM_QUERYITEMCOUNT - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

sresult ([SHORT](#))
Item count.

MM_QUERYITEMCOUNT - Syntax

This message returns the number of items in the menu.

```
param1
    ULONG    ulReserved /* Reserved value, should be 0. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */
```

MM_QUERYITEMCOUNT - Remarks

The menu control window procedure responds to this message by returning the count of the number of items in the menu.

MM_QUERYITEMCOUNT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *result* to the default value of 0.

MM_QUERYITEMCOUNT - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

MM_QUERYITEMRECT

MM_QUERYITEMRECT Field - usitem

usitem (USHORT)
Item identity.

MM_QUERYITEMRECT Field - flIncludeSubmenus

flIncludeSubmenus (BOOL)
Include submenus indicator.

- TRUE If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and return its state.
- FALSE If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

MM_QUERYITEMRECT Field - prect

prect (PRECTL)
Bounding rectangle of the menu item in device coordinates relative to the menu window.

MM_QUERYITEMRECT Return Value - rc

- rc** (BOOL)
Success indicator.
- TRUE Specified item was found.
- FALSE Specified item was not found.

MM_QUERYITEMRECT - Parameters

usitem (USHORT)
Item identity.

flIncludeSubmenus (BOOL)
Include submenus indicator.

- TRUE If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and return its state.
- FALSE

If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

prect ([PRECTL](#))
Bounding rectangle of the menu item in device coordinates relative to the menu window.

rc ([BOOL](#))
Success indicator.

TRUE	Specified item was found.
FALSE	Specified item was not found.

MM_QUERYITEMRECT - Syntax

This message returns the bounding rectangle of a menu item.

```
param1
    USHORT  usitem          /* Item identity. */
    BOOL     fIncludeSubmenus /* Include submenus indicator. */

param2
    PRECTL  prect           /* Bounding rectangle of the menu item in device coordinates relative to the me
```

MM_QUERYITEMRECT - Remarks

The menu control responds to this message by returning the bounding rectangle of identified menu item.

MM_QUERYITEMRECT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of 0 (FALSE).

MM_QUERYITEMRECT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MM_QUERYITEMTEXT

MM_QUERYITEMTEXT Field - usitem

usitem (USHORT)
Item identifier.

MM_QUERYITEMTEXT Field - smaxcount

smaxcount (SHORT)
Maximum count.

Copy the item text as a null-terminated string, but limit the number of characters copied, including the null termination character, to this value, which must be greater than 0.

MM_QUERYITEMTEXT Field - pszItemText

pszItemText (PSZ)
Buffer into which the item text is to be copied.

This points to a string (character) buffer.

MM_QUERYITEMTEXT Return Value - sTextLength

sTextLength (SHORT)
Length of item text.

The length of the text string, excluding the null termination character.

- 0
Error occurred. For example, no item of the specified identity exists or the item has no text. No text is copied.
- Other
Length of item text.

MM_QUERYITEMTEXT - Parameters

usitem ([USHORT](#))
Item identifier.

smaxcount ([SHORT](#))
Maximum count.

Copy the item text as a null-terminated string, but limit the number of characters copied, including the null termination character, to this value, which must be greater than 0.

pszItemText ([PSZ](#))
Buffer into which the item text is to be copied.

This points to a string (character) buffer.

sTextLength ([SHORT](#))
Length of item text.

The length of the text string, excluding the null termination character.

0
Error occurred. For example, no item of the specified identity exists or the item has no text. No text is copied.

Other
Length of item text.

MM_QUERYITEMTEXT - Syntax

This message returns the text of the specified menu item.

```
param1
    USHORT  usitem      /* Item identifier. */
    SHORT   smaxcount   /* Maximum count. */

param2
    PSZ     pszItemText /* Buffer into which the item text is to be copied. */
```

MM_QUERYITEMTEXT - Remarks

The menu control window procedure responds to this message by copying up to *smaxcount* characters as a null-terminated string from the text of the item specified by *usitem*, if it has the style MIS_TEXT, into the buffer specified by *pszItemText*.

The length of the item text can be determined by using the [MM_QUERYITEMTEXTLENGTH](#) message.

MM_QUERYITEMTEXT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sTextLength* to

the default value of 0.

MM_QUERYITEMTEXT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MM_QUERYITEMTEXTLENGTH

MM_QUERYITEMTEXTLENGTH Field - usitem

usitem ([USHORT](#))
Item identifier.

MM_QUERYITEMTEXTLENGTH Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MM_QUERYITEMTEXTLENGTH Return Value - sLength

sLength ([SHORT](#))
Length of item text.

The length of the text string, excluding the null termination character.

0
Error occurred. For example, no item of the specified identity exists or the item has no text. No text is copied.

Other
Length of item text.

MM_QUERYITEMTEXTLENGTH - Parameters

usitem (**USHORT**)
Item identifier.

ulReserved (**ULONG**)
Reserved value, should be 0.

sLength (**SHORT**)
Length of item text.

The length of the text string, excluding the null termination character.

0
Error occurred. For example, no item of the specified identity exists or the item has no text. No text is copied.

Other
Length of item text.

MM_QUERYITEMTEXTLENGTH - Syntax

This message returns the text length of the specified menu item.

```
param1
    USHORT  usitem      /* Item identifier. */

param2
    ULONG   ulReserved /* Reserved value, should be 0. */
```

MM_QUERYITEMTEXTLENGTH - Remarks

The menu control window procedure responds to this message by returning the length in characters of the text of the identified item, if it has a style of MIS_TEXT.

MM_QUERYITEMTEXTLENGTH - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sLength* to the default value of 0.

MM_QUERYITEMTEXTLENGTH - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

MM_QUERYSELITEMID

MM_QUERYSELITEMID Field - usReserve

usReserve ([USHORT](#))
Reserved value, should be 0.

MM_QUERYSELITEMID Field - usincludesubmenus

usincludesubmenus ([USHORT](#))
Include submenus indicator.

TRUE	If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for a selected item with the specified identifier.
FALSE	If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for a selected item with the specified identifier.

MM_QUERYSELITEMID Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MM_QUERYSELITEMID Return Value - sresult

sresult ([SHORT](#))
Selected item identifier.

MID_ERROR	Error occurred
MIT_NONE	No item selected
Other	Selected item identifier.

MM_QUERYSELITEMID - Parameters

usReserve ([USHORT](#))
Reserved value, should be 0.

usincludesubmenus ([USHORT](#))
Include submenus indicator.

TRUE	If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for a selected item with the specified identifier.
FALSE	If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for a selected item with the specified identifier.

ulReserved ([ULONG](#))
Reserved value, should be 0.

sresult ([SHORT](#))
Selected item identifier.

MID_ERROR	Error occurred
MIT_NONE	No item selected
Other	Selected item identifier.

MM_QUERYSELITEMID - Syntax

This message returns the identity of the selected menu item.

```
param1
    USHORT  usReserve           /* Reserved value, should be 0. */
    USHORT  usincludesubmenus  /* Include submenus indicator. */

param2
    ULONG   ulReserved         /* Reserved value, should be 0. */
```

MM_QUERYSELITEMID - Remarks

The menu control window procedure responds to this message by returning the identity of the selected item in the menu. Submenus and subdialogs are not searched unless *usincludesubmenus* is set to TRUE.

MM_QUERYSELITEMID - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sresult* to the default value of 0.

MM_QUERYSELITEMID - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

MM_REMOVEITEM

MM_REMOVEITEM Field - usitem

usitem ([USHORT](#))
Item identifier.

MM_REMOVEITEM Field - usincludesubmenus

usincludesubmenus ([USHORT](#))
Include submenu indicator.

- | | |
|-------|--|
| TRUE | If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and delete it. |
| FALSE | If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier. |

MM_REMOVEITEM Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MM_REMOVEITEM Return Value - slItemsLeft

slItemsLeft (SHORT)
Count of remaining items.

MM_REMOVEITEM - Parameters

usitem (USHORT)
Item identifier.

usincludesubmenus (USHORT)
Include submenus indicator.

- TRUE

If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and delete it.
- FALSE

If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

ulReserved (ULONG)
Reserved value, should be 0.

slItemsLeft (SHORT)
Count of remaining items.

MM_REMOVEITEM - Syntax

This message removes a menu item.

```
param1
    USHORT  usitem           /* Item identifier. */
    USHORT  usincludesubmenus /* Include submenus indicator. */

param2
    ULONG   ulReserved       /* Reserved value, should be 0. */
```

MM_REMOVEITEM - Remarks

The menu control window procedure responds to this message by removing the identified item from the menu and setting *sItemsLeft* to the count of items in the menu after the item is deleted.

The difference between this message and [MM_DELETEITEM](#) is that [MM_DELETEITEM](#) destroys any submenu window, and deletes any bit map associated with the item, whereas MM_REMOVEITEM does not.

Note: It must be sent, not posted, to the menu control.

MM_REMOVEITEM - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sItemsLeft* to the default value of 0.

MM_REMOVEITEM - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MM_SELECTITEM

MM_SELECTITEM Field - sitem

sitem (SHORT)	Item identifier.
MIT_NONE	Deselect all the items in the menu.
Other	Item identifier.

MM_SELECTITEM Field - usincludesubmenus

usincludesubmenus (USHORT)
Include submenus indicator.

- | | |
|-------|--|
| TRUE | If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and select or deselect it. |
| FALSE | If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier. |

MM_SELECTITEM Field - usReserve

usReserve (USHORT)
Reserved value, should be 0.

MM_SELECTITEM Field - usdismissed

usdismissed (USHORT)
Dismissed flag.

- | | |
|-------|--------------------------|
| TRUE | Dismiss the menu |
| FALSE | Do not dismiss the menu. |

MM_SELECTITEM Return Value - rc

rc (BOOL)
Success indicator.

- | | |
|-------|---|
| TRUE | A selection has been made, or <i>sitem</i> is MIT_NONE. |
| FALSE | A selection has not been made, or a deselection has been made, or <i>sitem</i> is not MIT_NONE. |

MM_SELECTITEM - Parameters

sitem (SHORT)	Item identifier.
MIT_NONE	Deselect all the items in the menu.
Other	Item identifier.
usincludesubmenus (USHORT)	Include submenus indicator.
TRUE	If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and select or deselect it.
FALSE	If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.
usReserve (USHORT)	Reserved value, should be 0.
usdismissed (USHORT)	Dismissed flag.
TRUE	Dismiss the menu
FALSE	Do not dismiss the menu.
rc (BOOL)	Success indicator.
TRUE	A selection has been made, or <i>sitem</i> is MIT_NONE.
FALSE	A selection has not been made, or a deselection has been made, or <i>sitem</i> is not MIT_NONE.

MM_SELECTITEM - Syntax

This message selects or deselects a menu item.

```
param1
    SHORT    sitem          /* Item identifier. */
    USHORT   usincludesubmenus /* Include submenus indicator. */

param2
    USHORT   usReserve      /* Reserved value, should be 0. */
    USHORT   usdismissed   /* Dismissed flag. */
```

MM_SELECTITEM - Remarks

The menu control window procedure responds to this message by setting the selection state of the (sub)menu which contains the specified item to indicate that the item is selected or deselected. If *usincludesubmenus* is set to TRUE, the selection state of the (sub)menu owning

the submenu which contains the specified item is also set. This process continues up the menu hierarchy until the top level menu is reached.

If an item is selected, and *usdismitted* is set to TRUE, a [WM_COMMAND](#), [WM_SYSCOMMAND](#), or [WM_HELP](#) message, as appropriate, is posted to the owner, and the menu is dismissed.

Note: This message must be sent, not posted, to the menu control.

MM_SELECTITEM - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

MM_SELECTITEM - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MM_SETDEFAULTITEMID

MM_SETDEFAULTITEMID Field - ulDeflItemID

ulDeflItemID ([ULONG](#))
The menu id of the item to become the new default.

MM_SETDEFAULTITEMID Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, must be 0.

MM_SETDEFAULTITEMID Return Value - rc

rc (BOOL)	Success of failure indicator.
TRUE	The conditional cascade default was set.
FALSE	The conditional cascade default was not set.

MM_SETDEFAULTITEMID - Parameters

ulDefItemID (ULONG)	The menu id of the item to become the new default.
ulReserved (ULONG)	Reserved value, must be 0.
rc (BOOL)	Success of failure indicator.
TRUE	The conditional cascade default was set.
FALSE	The conditional cascade default was not set.

MM_SETDEFAULTITEMID - Syntax

This message is used to set the default item in a conditional cascade menu.

```
param1
    ULONG  ulDefItemID

param2
    ULONG  ulReserved  /* Reserved value, must be 0. */
```

MM_SETDEFAULTITEMID - Remarks

The default item is the menu-id that will be returned if the main menu option is clicked on.

Open	(->)	Icon	id=MID_ICON
		*Tree	id=MID_TREE
		Details	id=MID_DETAILS

In the example above, where MID_TREE is currently the default, if the user clicked on the "Open" option without opening the conditional cascade menu, the menu would send back a notification that MID_TREE was selected.

MM_SETDEFAULTITEMID - Default Processing

The default window procedure takes no action other than to return 0.

MM_SETDEFAULTITEMID - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MM_SETITEM

MM_SETITEM Field - usReserve

usReserve ([USHORT](#))
Reserved value, should be 0.

MM_SETITEM Field - usincludesubmenus

usincludesubmenus ([USHORT](#))
Include submenus indicator.

- | | |
|-------|---|
| TRUE | If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and set its definition. |
| FALSE | If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier. |

MM_SETITEM Field - pmenuitem

pmenuitem (PMENUITEM)
Menu-item data structure.

This points to a MENUITEM structure.

MM_SETITEM Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

MM_SETITEM - Parameters

usReserve (USHORT)
Reserved value, should be 0.

usincludesubmenus (USHORT)
Include submenus indicator.

TRUE	If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and set its definition.
FALSE	If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

pmenuitem (PMENUITEM)
Menu-item data structure.

This points to a MENUITEM structure.

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

MM_SETITEM - Syntax

This message sets the definition of a menu item.

```
param1
    USHORT    usReserve      /* Reserved value, should be 0. */
    USHORT    usincludesubmenus /* Include submenus indicator. */

param2
    PMENUITEM pmenuitem      /* Menu-item data structure. */
```

MM_SETITEM - Remarks

The menu control window procedure responds to this message by using the specified structure to update the definition of the identified menu item.

The *iPosition* field of the structure specified by *pmenuitem* is ignored, as the position of the item cannot be changed by use of this message.

Note: It must be sent, not posted, to the menu control.

MM_SETITEM - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

MM_SETITEM - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MM_SETITEMATTR

MM_SETITEMATTR Field - usitem

usitem (USHORT)
Item identifier.

MM_SETITEMATTR Field - usincludesubmenus

usincludesubmenus (USHORT)
Include submenus indicator.

- TRUE

If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and set its attributes.
- FALSE

If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

MM_SETITEMATTR Field - usattributemask

usattributemask (USHORT)
Attribute mask.

MM_SETITEMATTR Field - usattributedata

usattributedata (USHORT)
Attribute data.

MM_SETITEMATTR Return Value - rc

- rc** (BOOL)
Success indicator.
- TRUE

Successful completion
- FALSE

Error occurred.

MM_SETITEMATTR - Parameters

usitem (USHORT)
Item identifier.

usincludesubmenus (USHORT)
Include submenus indicator.

TRUE
If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and set its attributes.

FALSE
If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

usattributemask (USHORT)
Attribute mask.

usattributedata (USHORT)
Attribute data.

rc (BOOL)
Success indicator.

TRUE
Successful completion

FALSE
Error occurred.

MM_SETITEMATTR - Syntax

This message sets the attributes of a menu item.

```
param1
    USHORT  usitem           /* Item identifier. */
    USHORT  usincludesubmenus /* Include submenus indicator. */

param2
    USHORT  usattributemask  /* Attribute mask. */
    USHORT  usattributedata /* Attribute data. */
```

MM_SETITEMATTR - Remarks

The menu control window procedure responds to this message by setting the state of the specified attributes for the identified item.

Note: It must be sent, not posted, to the menu control.

MM_SETITEMATTR - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

MM_SETITEMATTR - Examples

This example sends an MM_SETITEMATTR message to set the IDM_LARGE menu item's state to checked, and then sends another MM_SETITEMATTR message to set the IDM_MEDIUM menu item's state to unchecked.

```
WinSendMsg(hwndActionBar, MM_SETITEMATTR,
    MPFROM2SHORT(IDM_LARGE, TRUE),
    MPFROM2SHORT(MIA_CHECKED, MIA_CHECKED));
WinSendMsg(hwndActionBar, MM_SETITEMATTR,
    MPFROM2SHORT(IDM_MEDIUM, TRUE),
    MPFROM2SHORT(MIA_CHECKED, FALSE));
```

MM_SETITEMATTR - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Examples](#)
[Glossary](#)

MM_SETITEMHANDLE

MM_SETITEMHANDLE Field - usitem

usitem ([USHORT](#))
Item ID.

MM_SETITEMHANDLE Field - ulitemhandle

ulitemhandle ([ULONG](#))
Item handle.

MM_SETITEMHANDLE Return Value - rc

rc (BOOL)

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

MM_SETITEMHANDLE - Parameters

usitem (USHORT)

Item ID.

ulitemhandle (ULONG)

Item handle.

rc (BOOL)

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

MM_SETITEMHANDLE - Syntax

This message sets the handle of a menu item.

```
param1
    USHORT    usitem        /* Item ID. */

param2
    ULONG     ulitemhandle  /* Item handle. */
```

MM_SETITEMHANDLE - Remarks

The menu control window procedure responds to this message by setting the handle of the menu item.

This is used to set a handle for menu items that have a style of MIS_BITMAP or MIS_OWNERDRAW.

MM_SETITEMHANDLE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

MM_SETITEMHANDLE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

MM_SETITEMTEXT

MM_SETITEMTEXT Field - usitem

usitem ([USHORT](#))
Item identifier.

MM_SETITEMTEXT Field - pszItemText

pszItemText ([PSZ](#))
Item text.

This points to a string containing the text to set the menu item to.

MM_SETITEMTEXT Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

MM_SETITEMTEXT - Parameters

usitem (**USHORT**)
Item identifier.

pszItemText (**PSZ**)
Item text.

This points to a string containing the text to set the menu item to.

rc (**BOOL**)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

MM_SETITEMTEXT - Syntax

This message sets the text of a menu item.

```
param1
    USHORT  usitem          /* Item identifier. */
param2
    PSZ     pszItemText     /* Item text. */
```

MM_SETITEMTEXT - Remarks

The menu control responds to this message by setting the text of the identified item, if it has a style of `MIS_TEXT`, using the specified null-terminated string.

To right-align text in a menu item, you must imbed an ASCII 8 (0x08) character preceding it.

Note: This message must be sent, not posted, to the menu control.

MM_SETITEMTEXT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

MM_SETITEMTEXT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MM_STARTMENU MODE

MM_STARTMENU MODE Field - usshowsubmenu

usshowsubmenu (USHORT)	
Show submenu flag.	
TRUE	Show the submenu (pull-down menu) of the selected action bar item when the menu enters selection mode. If the action bar is not visible, the submenu is shown, otherwise it is not shown. If the item selected does not have a submenu, this parameter is ignored.
FALSE	Do not show the submenu (pull-down menu) of the selected action bar item when the menu enters selection mode.

MM_STARTMENU MODE Field - usresumemenu

usresumemenu (USHORT)	
Resume menu mode flag.	
TRUE	Resume the user interaction with the menu from where it left off. The menu is assumed to have been used previously and left without dismissing one of the submenus, and therefore is resumed in that submenu.
FALSE	Begin user interaction with the menu from the action bar, subject to the value of the <i>usshowsubmenu</i> parameter.

MM_STARTMENU MODE Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MM_STARTMENU MODE Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

MM_STARTMENU MODE - Parameters

usshowsubmenu (USHORT)
Show submenu flag.

TRUE	Show the submenu (pull-down menu) of the selected action bar item when the menu enters selection mode. If the action bar is not visible, the submenu is shown, otherwise it is not shown. If the item selected does not have a submenu, this parameter is ignored.
FALSE	Do not show the submenu (pull-down menu) of the selected action bar item when the menu enters selection mode.

usresumemenu (USHORT)
Resume menu mode flag.

TRUE	Resume the user interaction with the menu from where it left off. The menu is assumed to have been used previously and left without dismissing one of the submenus, and therefore is resumed in that submenu.
FALSE	Begin user interaction with the menu from the action bar, subject to the value of the <i>usshowsubmenu</i> parameter.

ulReserved (ULONG)
Reserved value, should be 0.

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

MM_STARTMENU MODE - Syntax

This message is used to begin menu selection.

```
param1
    USHORT  usshowsubmenu /* Show submenu flag. */
    USHORT  usresumemenu  /* Resume menu mode flag. */

param2
    ULONG    ulReserved    /* Reserved value, should be 0. */
```

MM_STARTMENU MODE - Remarks

It is posted to the menu when the operator presses the menu key.

Note: It must be posted, not sent, to the menu control.

MM_STARTMENU MODE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

MM_STARTMENU MODE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_QUERYCONVERTPOS (in Menu Controls)

WM_QUERYCONVERTPOS (in Menu Controls) - Syntax

For the cause of this message, see [WM_QUERYCONVERTPOS](#).

For a description of the parameters, see [WM_QUERYCONVERTPOS](#).

WM_QUERYCONVERTPOS (in Menu Controls) - Remarks

The menu control window procedure returns QCP_NOCONVERT.

WM_QUERYCONVERTPOS (in Menu Controls) - Default Processing

For the default window procedure processing of this message see [WM_QUERYCONVERTPOS](#).

WM_QUERYCONVERTPOS (in Menu Controls) - Related Messages

Related Messages

- [WM_QUERYCONVERTPOS](#)
-

WM_QUERYCONVERTPOS (in Menu Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_QUERYWINDOWPARAMS (in Menu Controls)

WM_QUERYWINDOWPARAMS (in Menu Controls) - Syntax

Occurs when an application queries the menu control window procedure parameters.

For a description of the parameters, see [WM_QUERYWINDOWPARAMS](#).

WM_QUERYWINDOWPARAMS (in Menu Controls) - Remarks

The menu control window procedure responds to this message by passing it to the default window procedure.

WM_QUERYWINDOWPARAMS (in Menu Controls) - Default Processing

The default window procedure sets the *cchText*, *cbPresParams*, and *cbCtlData* parameters of the [WNDPARAMS](#) data structure, identified by *pwndparams*, to 0 and sets *rc* to FALSE.

WM_QUERYWINDOWPARAMS (in Menu Controls) - Related Messages

Related Messages

- [WM_QUERYWINDOWPARAMS](#)
-

WM_QUERYWINDOWPARAMS (in Menu Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_SETWINDOWPARAMS (in Menu Controls)

WM_SETWINDOWPARAMS (in Menu Controls) - Syntax

This message occurs when an application sets or changes the menu control window procedure parameters.

For a description of the parameters, see [WM_SETWINDOWPARAMS](#).

WM_SETWINDOWPARAMS (in Menu Controls) - Remarks

The menu control window procedure responds to this message by passing it to the default window procedure.

WM_SETWINDOWPARAMS (in Menu Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_SETWINDOWPARAMS (in Menu Controls) - Related Messages

Related Messages

- [WM_SETWINDOWPARAMS](#)
-

WM_SETWINDOWPARAMS (in Menu Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_SYSCOMMAND

WM_SYSCOMMAND - Syntax

For the cause of this message, see [WM_SYSCOMMAND](#).

For a description of the parameters, see [WM_SYSCOMMAND](#).

The menu control window procedure sets *uscmd* to the menu-item identity.

WM_SYSCOMMAND - Remarks

The menu control window procedure generates this message and posts it to the queue of its owner, when an item is selected that has the style of MIS_SYSCOMMAND, but only if the [WM_MENUSELECT \(in Menu Controls\)](#) message returns a *rc* of TRUE.

WM_SYSCOMMAND - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_SYSCOMMAND - Topics

Select an item:

[Syntax](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

Multi-Line Entry Field Control Window Processing

This system-provided window procedure processes the actions on a multi-line entry field control (WC_MLE).

Purpose

A multi-line entry field control is a rectangular window that displays multiple lines of text that the operator can edit. When it has the focus, the cursor marks the current **insertion** or **replacement** point.

The text is displayed within a rectangular window. Scroll bars appear if requested.

On all four sides of the text within the window there exists a thin margin area. This margin remains drawn in the window's background color, and characters are never drawn into this margin. Mouse events that occur in the margin are processed differently from mouse events that occur in the text area. The margin should be large enough to be easily clicked on, but not so large as to take up a large quantity of screen space. It is suggested, but not required, that the left and right margins be half the average character width of the system font, and that the top and bottom margins be half the maximum baseline extent of the system font.

Text is defined as a stream of characters, with hard line-break characters in the text. Between any two bytes in the text stream, and at either end of the document, there is an insertion point. Note that in a DBCS environment, it is possible to have an insertion point in the middle of a DBCS character. If such an insertion point is specified in a function, the function will either round the insertion point in a sensible way, or the function will fail with an error code indicating the problem.

The text always contains a selection region, defined by an anchor point and a cursor point. The anchor and cursor points are insertion points. If the MLE window has the focus, the text between these two points is drawn highlighted and the cursor point is indicated by a flashing text cursor. The selection region can be affected by some import/export operations.

The cursor point and the anchor point define the range of the selection. These two points are often the same, in which case no text is selected and only a text cursor (but no highlighting) is displayed. A user can use SHIFT+cursor movement combinations to extend the selection, which leaves the anchor point alone, and moves the cursor point to a new position in the document.

The MLE has three modes:

INSERT/OVERTYPE

This mode determines whether keystrokes are inserted into the text, or whether they overwrite existing text. Unlike the other two modes, this mode is maintained by the system. The MLE must merely be aware of the system mode.

READ-ONLY

The keyboard user interface disallows any operations that would change the content of the text, although applications using the MLE can still change the text contents. The application can query this mode, in order that it can disallow application-specific operations.

WORD-WRAP

When this mode is in effect, soft line-breaks are inserted into the text at word boundaries so that the user need not scroll the display horizontally to see all the text. When this mode is off, text is allowed to trail off the right-hand edge of the window.

Notes:

1. The MLE is intended for text under 4Kb in size. Performance will be fast for text up to 32KB in size. Text greater than this will be supported but performance may not be acceptable.
2. In this section 'CR' denotes carriage-return, and 'LF' denotes line-feed.

Multi-Line Entry Field Control Data

See [MLECTLDATA](#) for multi-line entry field control data.

Multi-Line Entry Field Control Styles

These multi-line entry field control styles are available:

MLS_BORDER

Draws a border around the MLE field.

MLS_DISABLEUNDO

Directs the MLE control not to allow undo actions.

MLS_HSCROLL

Adds a horizontal scroll bar to the MLE field. The MLE control enables this scroll bar whenever any line exceeds the width of the MLE field.

MLS_IGNORETAB

Directs the MLE control to ignore the Tab key. It passes the appropriate [WM_CHAR](#) to its owner window.

MLS_LIMITVSCROLL

Displays the last MLE line at the bottom of the screen page. When this style is not used, the MLE control shows an empty space between the last MLE line and the bottom of the screen page.

MLS_READONLY

Prevents the MLE field from accepting text from the user. This style is useful for displaying lengthy static text in a client or dialog window.

MLS_VSCROLL

Adds a vertical scroll bar to the MLE field. The MLE control enables this scroll bar whenever the number of lines exceeds the height of the MLE field.

MLS_WORDWRAP

Automatically breaks lines that are longer than the width of the MLE field.

Multi-Line Entry Field Control Notification Messages

This message is initiated by the multi-line entry field window procedure to notify its owner of significant events.

WM_CONTROL (in Multiline Entry Fields)

WM_CONTROL (in Multiline Entry Fields) Field - usid

usid ([USHORT](#))
Control window identity.

WM_CONTROL (in Multiline Entry Fields) Field - usnotifycode

usnotifycode ([USHORT](#))
Notify code.

MLN_TEXTOVERFLOW

A key stroke causes the amount of text to exceed the limit on the number of bytes of data (refer to [MLM_SETTEXTLIMIT](#)). The parameter contains the number of bytes of data which would not fit within the current text limit. For character key strokes this can be 1 or 2 (DBCS). For Shift+Ins (paste) it can be any amount up to the paste limit.

The default *rc* of FALSE causes the default error handling, which is to ignore the key stroke, and beep.

An *rc* of TRUE implies that corrective action has been taken (such as deleting existing text or raising the limit) and the [WM_CHAR \(in Multiline Entry Fields\)](#) should be reprocessed as if just entered.

MLN_PIXHORZOVERFLOW

A key stroke causes the size of the display bit map to exceed the horizontal limit of the format rectangle (refer to [MLM_SETFORMATRECT](#)). The parameter contains the number of pels that would not fit within the current text limit.

The default *rc* of FALSE causes the default error handling, which is to ignore the key stroke, and beep.

An *rc* of TRUE implies that corrective action has been taken (such as changing to a smaller font or raising the limit) and the [WM_CHAR \(in Multiline Entry Fields\)](#) should be reprocessed as if just entered.

MLN_PIXVERTOVERFLOW

A key stroke causes the size of the display bit map to exceed the vertical limit of the format rectangle (refer to [MLM_SETFORMATRECT](#)). The parameter contains the number of pels that would not fit within the current text limit.

The default *rc* of FALSE causes the default error handling, which is to ignore the key stroke, and beep.

An *rc* of TRUE implies that corrective action has been taken (such as changing to a smaller font or raising the limit) and the [WM_CHAR \(in Multiline Entry Fields\)](#) should be reprocessed as if just entered.

MLN_OVERFLOW

An action other than entry of a key stroke causes a condition involving the text limit or format rectangle limit, such that either the limit becomes inadequate to contain the text or the text exceeds the limit.

This can be caused by:

- [MLM_SETWRAP](#)
- [MLM_SETTABSTOP](#)
- [MLM_SETFONT](#)
- [MLM_IMPORT](#)
- [MLM_PASTE](#)
- [MLM_CUT](#)
- [MLM_UNDO](#)
- [MLM_DELETE](#)

WM_SIZE.

MLN_HSCROLL

Indicates that the MLE has completed a scrolling calculation and is about to update the display accordingly. All queries return values as if the scrolling were complete. However, no scrolling action is visible on the user interface.

MLN_VSCROLL

Indicates that the MLE has completed a scrolling calculation and is about to update the display accordingly. All queries return values as if the scrolling were complete. However, no scrolling action is visible on the user interface.

MLN_CHANGE

Signals that the text has changed. This notification is sent whenever any text change occurs.

MLN_UNDOOVERFLOW

Signals that the text change operation, which could normally be undone, cannot be undone because the amount of text involved exceeds the undo capability. This includes text entry, deletion, cutting, and pasting.

MLN_CLPBDFAIL

Signals that a clipboard operation failed.

MLN_MEMERROR

Signals that the required storage cannot be obtained. The action that results in the increased storage requirement fails.

MLN_SETFOCUS

Sent whenever the MLE window receives the input focus.

MLN_KILLFOCUS

Sent whenever the MLE window loses the input focus.

MLN_MARGIN

Whenever the user moves the mouse into the left, right top, or bottom margins, this message is sent to the owner of the window.

If the owner returns an *rc* of TRUE, the mouse move is assumed to have been processed by the owner and no further action need be taken.

If the owner returns an *rc* of FALSE, the MLE performs a default action appropriate to each different mouse action.

The exceptions to this are all mouse messages that occur after a button-down inside the margin, until and including the matching button-up. Conceptually the drag (button-down until button-up) is a single macro event. Therefore, if FALSE is returned for a button-down event, no further margin notifications are given until after the drag has ended (button-up).

Note: If the application receives a notification of button-down in the margin and processes it, it must capture the mouse until the button-up event.

MLN_SEARCHPAUSE

This notification is sent periodically by the MLE, while an [MLM_SEARCH](#) message is being processed, to give an application the opportunity to stop excessively long searches, and to provide search progress information. The owner window can respond either with TRUE or FALSE. FALSE causes the MLE to continue searching; TRUE causes the MLE to stop the search immediately. For further information, see [MLM_SEARCH](#)

WM_CONTROL (in Multiline Entry Fields) Field - ulOver

ulOver (ULONG)

Number of bytes that do not fit.

param2 contains *ulOver* for a *usnotifycode* of MLN_TEXTOVERFLOW.

WM_CONTROL (in Multiline Entry Fields) Field - pixOver

pixOver (PIX)

Linear distance of overflow in pels.

param2 contains *pixOver* for a *usnotifycode* of MLN_PIXHORZOVERFLOW or MLN_PIXVERTOVERFLOW.

WM_CONTROL (in Multiline Entry Fields) Field - pErrInfo

pErrInfo (POVERFLOW)

Overflow error information structure.

param2 contains *pErrInfo* for a *usnotifycode* of MLN_OVERFLOW.

The *afErrInd* field of the **MLEOVERFLOW** structure can take one or more of the following values:

MLFEFR_RESIZE

The window is resized, and the format rectangle is tied to the window size and limited either horizontally, vertically, or both. The implicit change of the format rectangle to the new size does not contain the text. The format rectangle is made static at the previous size, and the MLESFR_MATCHWINDOW style is turned off until set again by the application. This is done in response to a **WM_SIZE** message, and therefore the multi-line entry field does not forward the return value from this notification message.

MLFEFR_TABSTOP

A tab stop location change is requested, and the text is limited either horizontally, vertically, or both. Changing the tab stops causes the text to exceed the limit. The tab stop change is rejected.

MLFEFR_FONT

A font change is requested, and the text is limited either horizontally, vertically, or both. Changing the font causes the text to exceed the limit. The font change is rejected.

MLFEFR_WORDWRAP

The word-wrap state is requested to be changed, and the text is limited either horizontally, vertically, or both. Wrapping the text differently exceeds the limit, and the request is rejected. This happens in situations where the horizontal limit is not set, there are lines exceeding it, and word-wrap is being changed from off to on, such that it creates soft line breaks resulting in increased vertical size. This happens if word-wrap is being changed from on to off, and there is at least one line created by a soft line-break, such that when that line-break is removed, the full line (up to the hard line break) exceeds the horizontal limit.

MLFEFR_TEXT

Text is changed by **MLM_IMPORT**, **MLM_PASTE**, **MLM_CUT**, **MLM_UNDO**, or **MLM_DELETE**, and the text is limited either horizontally, vertically, or both within the format rectangle. The change causes the text to exceed the format rectangle in a dimension that is limited. For example, Delete and EOL joins text from two lines into one line long enough to exceed the horizontal limit.

MLFETL_TEXTBYTES

Text is changed by **MLM_IMPORT**, **MLM_PASTE**, or **MLM_UNDO**, and the text is limited to a maximum number of bytes. The change causes the text to exceed that maximum.

WM_CONTROL (in Multiline Entry Fields) Field - ulErrInd

ulErrInd (ULONG)

Clipboard fail flag.

param2 contains *ulErrInd* for a *usnotifycode* of MLN_CLPBDFAIL.

MLFCPBD_TOOMUCHTEXT

Text amount exceeds clipboard capacity
MLFCPBD_CLPBDERROR
A clipboard error occurred.

WM_CONTROL (in Multiline Entry Fields) Field - pmrg

pmrg ([PMARGSTRUCT](#))
Margin structure.

param2 contains *pmrg* for a *usnotifycode* of MLN_MARGIN.

The left and right margins are defined as going all the way to the top and bottom such that the top and bottom margins are contained between them. Therefore, the corners are included in the sides.

usMouMsg contains the mouse message that signals the event.

iptNear contains the insertion point of the nearest point in the text. For situations where the nearest location is beyond the end of a line, the insertion point for the end of the line is returned. (The EOL character is considered to be beyond the end of the line.)

WM_CONTROL (in Multiline Entry Fields) Field - iptSearchedTo

iptSearchedTo ([IPT](#))
Current insertion point of search.

param2 contains *iptSearchedTo* for a *usnotifycode* of MLN_SEARCHPAUSE.

WM_CONTROL (in Multiline Entry Fields) Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

param2 contains *ulReserved* for a *usnotifycode* of MLN_HSCROLL, MLN_VSCROLL, MLN_CHANGE, MLN_UNDOOVERFLOW, MLN_MEMERROR, MLN_SETFOCUS, or MLN_KILLFOCUS.

WM_CONTROL (in Multiline Entry Fields) Field - rc

rc ([BOOL](#))
Action taken by application.

ReturnCode contains *rc* for a *usnotifycode* of MLN_TEXTOVERFLOW, MLN_PIXHORZOVERFLOW, MLN_PIXVERTOVERFLOW, MLN_MARGIN, or MLN_SEARCHPAUSE.

TRUE

The multiline entry field control assumes that appropriate action has been taken by the application. Appropriate action depends on the MLN_* notification code, and is documented under the *usnotifycode* field.

FALSE

The multiline entry field control assumes that the application has ignored this WM_CONTROL (in Multiline Entry Fields) message, and takes action appropriate to the MLN_* notification code, as documented under the *usnotifycode* field.

WM_CONTROL (in Multiline Entry Fields) Field - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

ReturnCode contains *ulReserved* for a *usnotifycode* of MLN_OVERFLOW, MLN_HSCROLL, MLN_VSCROLL, MLN_CHANGE, MLN_UNDOOVERFLOW, MLN_CLPBDFAIL, MLN_MEMERROR, MLN_SETFOCUS, or MLN_KILLFOCUS.

WM_CONTROL (in Multiline Entry Fields) - Parameters

usid (USHORT)

Control window identity.

usnotifycode (USHORT)

Notify code.

MLN_TEXTOVERFLOW

A key stroke causes the amount of text to exceed the limit on the number of bytes of data (refer to [MLM_SETTEXTLIMIT](#)). The parameter contains the number of bytes of data which would not fit within the current text limit. For character key strokes this can be 1 or 2 (DBCS). For Shift+Ins (paste) it can be any amount up to the paste limit.

The default *rc* of FALSE causes the default error handling, which is to ignore the key stroke, and beep.

An *rc* of TRUE implies that corrective action has been taken (such as deleting existing text or raising the limit) and the [WM_CHAR \(in Multiline Entry Fields\)](#) should be reprocessed as if just entered.

MLN_PIXHORZOVERFLOW

A key stroke causes the size of the display bit map to exceed the horizontal limit of the format rectangle (refer to [MLM_SETFORMATRECT](#)). The parameter contains the number of pels that would not fit within the current text limit.

The default *rc* of FALSE causes the default error handling, which is to ignore the key stroke, and beep.

An *rc* of TRUE implies that corrective action has been taken (such as changing to a smaller font or raising the limit) and the [WM_CHAR \(in Multiline Entry Fields\)](#) should be reprocessed as if just entered.

MLN_PIXVERTOVERFLOW

A key stroke causes the size of the display bit map to exceed the vertical limit of the format rectangle (refer to [MLM_SETFORMATRECT](#)). The parameter contains the number of pels that would not fit within the current text limit.

The default *rc* of FALSE causes the default error handling, which is to ignore the key stroke, and beep.

An *rc* of TRUE implies that corrective action has been taken (such as changing to a smaller font or raising the limit) and the [WM_CHAR \(in Multiline Entry Fields\)](#) should be reprocessed as if just entered.

MLN_OVERFLOW

An action other than entry of a key stroke causes a condition involving the text limit or format rectangle limit, such that either the limit becomes inadequate to contain the text or the text exceeds the limit.

This can be caused by:

[MLM_SETWRAP](#)
[MLM_SETTABSTOP](#)
[MLM_SETFONT](#)
[MLM_IMPORT](#)
[MLM_PASTE](#)
[MLM_CUT](#)
[MLM_UNDO](#)
[MLM_DELETE](#)
[WM_SIZE](#).

MLN_HSCROLL

Indicates that the MLE has completed a scrolling calculation and is about to update the display accordingly. All queries return values as if the scrolling were complete. However, no scrolling action is visible on the user interface.

MLN_VSCROLL

Indicates that the MLE has completed a scrolling calculation and is about to update the display accordingly. All queries return values as if the scrolling were complete. However, no scrolling action is visible on the user interface.

MLN_CHANGE

Signals that the text has changed. This notification is sent whenever any text change occurs.

MLN_UNDOOVERFLOW

Signals that the text change operation, which could normally be undone, cannot be undone because the amount of text involved exceeds the undo capability. This includes text entry, deletion, cutting, and pasting.

MLN_CLPBDFAIL

Signals that a clipboard operation failed.

MLN_MEMERROR

Signals that the required storage cannot be obtained. The action that results in the increased storage requirement fails.

MLN_SETFOCUS

Sent whenever the MLE window receives the input focus.

MLN_KILLFOCUS

Sent whenever the MLE window loses the input focus.

MLN_MARGIN

Whenever the user moves the mouse into the left, right top, or bottom margins, this message is sent to the owner of the window.

If the owner returns an *rc* of TRUE, the mouse move is assumed to have been processed by the owner and no further action need be taken.

If the owner returns an *rc* of FALSE, the MLE performs a default action appropriate to each different mouse action.

The exceptions to this are all mouse messages that occur after a button-down inside the margin, until and including the matching button-up. Conceptually the drag (button-down until button-up) is a single macro event. Therefore, if FALSE is returned for a button-down event, no further margin notifications are given until after the drag has ended (button-up).

Note: If the application receives a notification of button-down in the margin and processes it, it must capture the mouse until the button-up event.

MLN_SEARCHPAUSE

This notification is sent periodically by the MLE, while an [MLM_SEARCH](#) message is being processed, to give an application the opportunity to stop excessively long searches, and to provide search progress information. The owner window can respond either with TRUE or FALSE. FALSE causes the MLE to continue searching; TRUE causes the MLE to stop the search immediately. For further information, see [MLM_SEARCH](#)

ulOver (ULONG)

Number of bytes that do not fit.

param2 contains *ulOver* for a *usnotifycode* of MLN_TEXTOVERFLOW.

pixOver (PIX)

Linear distance of overflow in pels.

param2 contains *pixOver* for a *usnotifycode* of MLN_PIXHORZOVERFLOW or MLN_PIXVERTOVERFLOW.

pErrInfo (POVERFLOW)

Overflow error information structure.

param2 contains *pErrInfo* for a *usnotifycode* of MLN_OVERFLOW.

The *afErrInd* field of the **MLEOVERFLOW** structure can take one or more of the following values:

MLFEFR_RESIZE

The window is resized, and the format rectangle is tied to the window size and limited either horizontally, vertically, or both. The implicit change of the format rectangle to the new size does not contain the text. The format rectangle is made static at the previous size, and the MLESFR_MATCHWINDOW style is turned off until set again by the application. This is done in response to a **WM_SIZE** message, and therefore the multi-line entry field does not forward the return value from this notification message.

MLFEFR_TABSTOP

A tab stop location change is requested, and the text is limited either horizontally, vertically, or both. Changing the tab stops causes the text to exceed the limit. The tab stop change is rejected.

MLFEFR_FONT

A font change is requested, and the text is limited either horizontally, vertically, or both. Changing the font causes the text to exceed the limit. The font change is rejected.

MLFEFR_WORDWRAP

The word-wrap state is requested to be changed, and the text is limited either horizontally, vertically, or both. Wrapping the text differently exceeds the limit, and the request is rejected. This happens in situations where the horizontal limit is not set, there are lines exceeding it, and word-wrap is being changed from off to on, such that it creates soft line breaks resulting in increased vertical size. This happens if word-wrap is being changed from on to off, and there is at least one line created by a soft line-break, such that when that line-break is removed, the full line (up to the hard line break) exceeds the horizontal limit.

MLFEFR_TEXT

Text is changed by **MLM_IMPORT**, **MLM_PASTE**, **MLM_CUT**, **MLM_UNDO**, or **MLM_DELETE**, and the text is limited either horizontally, vertically, or both within the format rectangle. The change causes the text to exceed the format rectangle in a dimension that is limited. For example, Delete and EOL joins text from two lines into one line long enough to exceed the horizontal limit.

MLFETL_TEXTBYTES

Text is changed by **MLM_IMPORT**, **MLM_PASTE**, or **MLM_UNDO**, and the text is limited to a maximum number of bytes. The change causes the text to exceed that maximum.

ulErrInd (ULONG)

Clipboard fail flag.

param2 contains *ulErrInd* for a *usnotifycode* of MLN_CLPBDFAIL.

MLFCPBD_TOOMUCHTEXT

Text amount exceeds clipboard capacity

MLFCPBD_CLPBDERROR

A clipboard error occurred.

pmrg (PMARGSTRUCT)

Margin structure.

param2 contains *pmrg* for a *usnotifycode* of MLN_MARGIN.

The left and right margins are defined as going all the way to the top and bottom such that the top and bottom margins are contained between them. Therefore, the corners are included in the sides.

usMouMsg contains the mouse message that signals the event.

iptNear contains the insertion point of the nearest point in the text. For situations where the nearest location is beyond the end of a line, the insertion point for the end of the line is returned. (The EOL character is considered to be beyond the end of the line.)

iptSearchedTo (IPT)

Current insertion point of search.

param2 contains *iptSearchedTo* for a *usnotifycode* of MLN_SEARCHPAUSE.

ulReserved (ULONG)

Reserved value, should be 0.

param2 contains *ulReserved* for a *usnotifycode* of MLN_HSCROLL, MLN_VSCROLL, MLN_CHANGE, MLN_UNDOOVERFLOW, MLN_MEMERROR, MLN_SETFOCUS, or MLN_KILLFOCUS.

rc (BOOL)

Action taken by application.

ReturnCode contains *rc* for a *usnotifycode* of MLN_TEXTOVERFLOW, MLN_PIXHORZOVERFLOW, MLN_PIXVERTOVERFLOW, MLN_MARGIN, or MLN_SEARCHPAUSE.

TRUE

The multiline entry field control assumes that appropriate action has been taken by the application. Appropriate action depends on the MLN_* notification code, and is documented under the *usnotifycode* field.

FALSE

The multiline entry field control assumes that the application has ignored this WM_CONTROL (in Multiline Entry Fields) message, and takes action appropriate to the MLN_* notification code, as documented under the *usnotifycode* field.

ulReserved (ULONG)

Reserved value, should be 0.

ReturnCode contains *ulReserved* for a *usnotifycode* of MLN_OVERFLOW, MLN_HSCROLL, MLN_VSCROLL, MLN_CHANGE, MLN_UNDOOVERFLOW, MLN_CLPBDFAIL, MLN_MEMERROR, MLN_SETFOCUS, or MLN_KILLFOCUS.

WM_CONTROL (in Multiline Entry Fields) - Syntax

For the cause of this message, see [WM_CONTROL](#).

```
param1
    USHORT    usid          /* Control window identity. */
    USHORT    usnotifycode  /* Notify code. */

param2
    ULONG     ulOver        /* Number of bytes that do not fit. */
    PIX       pixOver       /* Linear distance of overflow in pels. */
    POVERFLOW pErrInfo      /* Overflow error information structure. */
    ULONG     ulErrInd       /* Clipboard fail flag. */
    PMARGSTRUCT pmrg        /* Margin structure. */
    IPT       iptSearchedTo  /* Current insertion point of search. */
    ULONG     ulReserved    /* Reserved value, should be 0. */

returns
    BOOL      rc            /* Action taken by application. */
    ULONG     ulReserved    /* Reserved value, should be 0. */
```

WM_CONTROL (in Multiline Entry Fields) - Remarks

The multiline entry field control window procedure generates this message and sends it to its owner, informing the owner of the event.

param2 depends on the MLN_* notification code.

WM_CONTROL (in Multiline Entry Fields) - Default

Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_CONTROL (in Multiline Entry Fields) - Related Messages

Related Messages

- [WM_CONTROL](#)
-

WM_CONTROL (in Multiline Entry Fields) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

Multi-Line Entry Field Window Messages

This section describes the multi-line entry field control window procedure actions on receiving the following messages.

MLM_CHARFROMLINE

MLM_CHARFROMLINE Field - ILineNum

ILineNum ([LONG](#))
Line number of interest.

MLM_CHARFROMLINE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_CHARFROMLINE Return Value - iptFirst

iptFirst ([IPT](#))
First insertion point on line.

MLM_CHARFROMLINE - Parameters

lLineNum ([LONG](#))
Line number of interest.

ulReserved ([ULONG](#))
Reserved value, should be 0.

iptFirst ([IPT](#))
First insertion point on line.

MLM_CHARFROMLINE - Syntax

This message returns the first insertion point on a given line.

```
param1
    LONG    lLineNum    /* Line number of interest. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */
```

MLM_CHARFROMLINE - Remarks

For any line number, the insertion point just before the first character on that line is returned. If the line number is -1, the line containing the cursor is used.

The term line means a line on the display after the application of word-wrap. It does not mean a line as defined by the CR LF line-break sequence.

MLM_CHARFROMLINE - Default Processing

The default window procedure takes no action on this message, other than to set *ipFirst* to 0.

MLM_CHARFROMLINE - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

MLM_CLEAR

MLM_CLEAR Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_CLEAR Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_CLEAR Return Value - ulClear

ulClear ([ULONG](#))
Number of bytes deleted, counted in CF_TEXT format.

MLM_CLEAR - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulClear ([ULONG](#))
Number of bytes deleted, counted in CF_TEXT format.

MLM_CLEAR - Syntax

This message clears the current selection.

```
param1
    ULONG  ulReserved  /* Reserved value, should be 0. */

param2
    ULONG  ulReserved  /* Reserved value, should be 0. */
```

MLM_CLEAR - Remarks

The multi-line entry field control window procedure responds to this message by clearing the current selection and returning the number of bytes cleared.

MLM_CLEAR - Default Processing

The default window procedure takes no action on this message, other than to set *ulClear* to 0.

MLM_CLEAR - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MLM_COPY

MLM_COPY Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MLM_COPY Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MLM_COPY Return Value - ulCopy

ulCopy (ULONG)
Number of bytes transferred, counted in CF_TEXT format.

MLM_COPY - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

ulCopy (ULONG)
Number of bytes transferred, counted in CF_TEXT format.

MLM_COPY - Syntax

This message copies the current selection to the clipboard.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */
```

param2
 ULONG **ulReserved** /* Reserved value, should be 0. */

MLM_COPY - Remarks

The multi-line entry field control window procedure responds to this message by copying the selected text to the clipboard. The text is translated to standard clipboard format, which is the same as exporting with MLE_CFTTEXT format.

The text is placed on the clipboard as a single contiguous data segment. This restricts the amount to the maximum segment size (64KB).

This may cause an overflow, see MLN_OVERFLOW.

MLM_COPY - Default Processing

The default window procedure takes no action on this message, other than to set *ulCopy* to 0.

MLM_COPY - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

MLM_CUT

MLM_CUT Field - ulReserved

ulReserved (**ULONG**)
 Reserved value, should be 0.

MLM_CUT Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MLM_CUT Return Value - ulCopy

ulCopy (ULONG)
Number of bytes transferred, counted in CF_TEXT format.

MLM_CUT - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

ulCopy (ULONG)
Number of bytes transferred, counted in CF_TEXT format.

MLM_CUT - Syntax

This message copies the text that forms the current selection to the clipboard and then deletes it from the MLE control.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

MLM_CUT - Remarks

The multi-line entry field control window procedure responds to this message by copying the selected text to the clipboard and then deleting it. The text is translated to standard clipboard format, which is the same as exporting with MLE_CFTTEXT format.

The text is placed on the clipboard as a single contiguous data segment. This restricts the amount to the maximum segment size (64KB).

This may cause an overflow, see MLN_OVERFLOW.

MLM_CUT - Default Processing

The default window procedure takes no action on this message, other than to set *ulCopy* to 0.

MLM_CUT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MLM_DELETE

MLM_DELETE Field - iptBegin

iptBegin ([IPT](#))
Starting point of deletion.

MLM_DELETE Field - ulDel

ulDel ([ULONG](#))
Number of bytes to delete.

MLM_DELETE Return Value - ulSuccess

ulSuccess ([ULONG](#))
Number of bytes successfully deleted.

MLM_DELETE - Parameters

iptBegin ([IPT](#))

Starting point of deletion.

ulDel ([ULONG](#))

Number of bytes to delete.

ulSuccess ([ULONG](#))

Number of bytes successfully deleted.

MLM_DELETE - Syntax

This message deletes text.

```
param1
    IPT    iptBegin    /* Starting point of deletion. */

param2
    ULONG  ulDel       /* Number of bytes to delete. */
```

MLM_DELETE - Remarks

This message takes an insertion point and a length, and deletes that number of characters from the text. If the insertion point is -1, the selection is used and the effect is identical to the [MLM_CLEAR](#) message.

This may cause an overflow, see [MLN_OVERFLOW](#).

MLM_DELETE - Default Processing

The default window procedure takes no action on this message, other than to set *ulSuccess* to 0.

MLM_DELETE - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

MLM_DISABLELREFRESH

MLM_DISABLELREFRESH Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MLM_DISABLELREFRESH Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MLM_DISABLELREFRESH Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

MLM_DISABLELREFRESH - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

rc (BOOL)
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

MLM_DISABLELREFRESH - Syntax

This message disables screen refresh.

```
param1
    ULONG  ulReserved /* Reserved value, should be 0. */

param2
    ULONG  ulReserved /* Reserved value, should be 0. */
```

MLM_DISABLELREFRESH - Remarks

This message disables screen refreshes. This allows an application to make changes throughout a document while avoiding unnecessary overhead caused by attempts to keep the screen display current. When an [MLM_ENABLELREFRESH](#) message is sent, the screen display is brought up to date with the contents of the text.

While refresh is disabled, mouse and keyboard messages are processed by beeping and ignoring them, except for mouse moves, which do not beep; the mouse pointer changes to the system standard wait symbol (a clock face).

MLM_DISABLELREFRESH - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

MLM_DISABLELREFRESH - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

MLM_ENABLELREFRESH

MLM_ENABLELREFRESH Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MLM_ENABLELREFRESH Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

0
Reserved value, 0.

MLM_ENABLELREFRESH Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	An error occurred.

MLM_ENABLELREFRESH - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

0
Reserved value, 0.

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	An error occurred.

MLM_ENABLELREFRESH - Syntax

This message enables screen refresh.

```
param1
    ULONG  ulReserved /* Reserved value, should be 0. */

param2
    ULONG  ulReserved /* Reserved value, should be 0. */
```

MLM_ENABLELREFRESH - Remarks

This message enables screen refreshes. This allows an application to make changes throughout a document while avoiding unnecessary overhead caused by attempts to keep the screen display current. When an MLM_ENABLELREFRESH message is sent, the screen display is brought up to date with the contents of the text.

While refresh is disabled, mouse and keyboard messages are processed by beeping and ignoring them, except for mouse moves, which do not beep; the mouse pointer changes to the system standard wait symbol (a clock face).

MLM_ENABLELREFRESH - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

MLM_ENABLELREFRESH - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MLM_EXPORT

MLM_EXPORT Field - pBegin

pBegin ([PIPT](#))

Starting point.

Updated to follow the last character exported.

MLM_EXPORT Field - pCopy

pCopy (**PULONG**)

Number of bytes being exported.

Decrement by the number of bytes actually exported.

MLM_EXPORT Return Value - ulSuccess

ulSuccess (**ULONG**)

Number of bytes successfully exported.

MLM_EXPORT - Parameters

pBegin (**PIPT**)

Starting point.

Updated to follow the last character exported.

pCopy (**PULONG**)

Number of bytes being exported.

Decrement by the number of bytes actually exported.

ulSuccess (**ULONG**)

Number of bytes successfully exported.

MLM_EXPORT - Syntax

This message exports text to a buffer.

```
param1
    PIPT    pBegin    /* Starting point. */

param2
    PULONG  pCopy     /* Number of bytes being exported. */
```

MLM_EXPORT - Remarks

This message takes an insertion point and length as parameters, and copies text, starting from that insertion point, into the buffer set by [MLM_SETIMPORTEXP](#). Text is in the format set by [MLM_FORMAT](#). If the insertion point is -1, the selection is used for both *pBegin* and *pCopy*.

On return, *pBegin* is updated to follow the last byte exported, and the number of bytes to be exported is decremented by the number actually exported. This is done to prepare those parameter values for the next export. The return value indicates the number of bytes actually put into the buffer. This number is less than, or equal to, the buffer size (see [MLM_SETIMPORTEXP](#)).

Note: All exports are done in full characters. Therefore, if either the length of the buffer or the number of bytes to be exported result in the last byte transferred being only half of a DBCS character, the MLE will *not* transfer that byte.

It returns the number of bytes placed in the export buffer.

MLM_EXPORT - Default Processing

The default window procedure takes no action on this message, other than to set *uiSuccess* to 0.

MLM_EXPORT - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

MLM_FORMAT

MLM_FORMAT Field - usFormat

usFormat ([USHORT](#))

Format to be used for import and export.

MLFIE_CFTEXT

Text format. Each line ends with a carriage-return/line-feed combination. Tab characters separate fields within a line. A NULL character signals the end of the data.

MLFIE_NOTRANS

Uses LF for line delineation, and guarantees that any text imported into the MLE in this format can be recovered in

exactly the same form on export.

MLFIE_WINFMT

(Windows MLE format.) On import, recognizes CR LF as denoting hard line-breaks, and ignores the sequence CR CR LF. On export, uses CR LF to denote a hard line-break and CR CR LF to denote a soft line-break caused by word-wrapping.

MLM_FORMAT Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

MLM_FORMAT Return Value - usFormat

usFormat ([USHORT](#))

Previous format value.

MLM_FORMAT - Parameters

usFormat ([USHORT](#))

Format to be used for import and export.

MLFIE_CFTXT

Text format. Each line ends with a carriage-return/line-feed combination. Tab characters separate fields within a line. A NULL character signals the end of the data.

MLFIE_NOTRANS

Uses LF for line delineation, and guarantees that any text imported into the MLE in this format can be recovered in exactly the same form on export.

MLFIE_WINFMT

(Windows MLE format.) On import, recognizes CR LF as denoting hard line-breaks, and ignores the sequence CR CR LF. On export, uses CR LF to denote a hard line-break and CR CR LF to denote a soft line-break caused by word-wrapping.

ulReserved ([ULONG](#))

Reserved value, should be 0.

usFormat ([USHORT](#))

Previous format value.

MLM_FORMAT - Syntax

This message sets the format to be used for buffer importing and exporting.

```
param1
    USHORT  usFormat    /* Format to be used for import and export. */

param2
    ULONG   ulReserved  /* Reserved value, should be 0. */
```

MLM_FORMAT - Remarks

The default format is MLFIE_CFTEXT.

The keyword MLFIE_RTF is reserved.

MLM_FORMAT - Default Processing

The default window procedure takes no action on this message, other than to set *usFormat* to 0.

MLM_FORMAT - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

MLM_IMPORT

MLM_IMPORT Field - pBegin

pBegin ([PIPT](#))

Insertion point.

Updated to insertion point following last insert.

MLM_IMPORT Field - ulCopy

ulCopy ([ULONG](#))
Number of bytes in buffer.

MLM_IMPORT Return Value - ulSuccess

ulSuccess ([ULONG](#))
Number of bytes successfully inserted.

MLM_IMPORT - Parameters

pBegin ([PIPT](#))
Insertion point.

Updated to insertion point following last insert.

ulCopy ([ULONG](#))
Number of bytes in buffer.

ulSuccess ([ULONG](#))
Number of bytes successfully inserted.

MLM_IMPORT - Syntax

This message imports text from a buffer.

```
param1
    PIPT    pBegin    /* Insertion point. */

param2
    ULONG   ulCopy    /* Number of bytes in buffer. */
```

MLM_IMPORT - Remarks

This message takes an insertion point and length as parameters. It assumes a buffer has been set using [MLM_SETIMPORTEXPORT](#), and inserts the contents of the buffer at the insertion point in the text. The contents are interpreted as being in the format set by [MLM_FORMAT](#). If the insertion point is -1, the cursor point is used.

The insertion point *pBegin* is updated by the MLE to the point after the last character imported. This provides the application with the location for the next import.

The return value indicates how many bytes were actually transferred.

All imports are done in full characters, therefore, if the number of bytes to be imported results in the last byte transferred being only half of a DBCS character, or part of a line-break sequence (CR LF or CR CR LF), the MLE does not transfer that byte. If the return value indicates that less than the full amount was transferred, a check must be made to determine if it is the beginning of a multi-byte sequence, and if so, the parts must be mated and imported as a whole.

This can cause an overflow, see MLN_OVERFLOW.

Note: The buffer is not zero-terminated; NULL characters can be inserted into the text.

MLM_IMPORT - Default Processing

The default window procedure takes no action on this message, other than to set *uiSuccess* to 0.

MLM_IMPORT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MLM_INSERT

MLM_INSERT Field - pchText

pchText ([PCHAR](#))
Null-terminated text string.

MLM_INSERT Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_INSERT Return Value - ulCount

ulCount ([ULONG](#))
Number of bytes actually inserted.

MLM_INSERT - Parameters

pchText ([PCHAR](#))
Null-terminated text string.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulCount ([ULONG](#))
Number of bytes actually inserted.

MLM_INSERT - Syntax

This message deletes the current selection and replaces it with a text string.

```
param1  
    PCHAR  pchText    /* Null-terminated text string. */  
  
param2  
    ULONG  ulReserved /* Reserved value, should be 0. */
```

MLM_INSERT - Remarks

This message inserts the text string at the current selection, deleting that selection in the same manner as typing at the keyboard would. The text string must be in CF_TEXT format (or one of the formats acceptable to [MLM_IMPORT](#)) and null-terminated. The line-break (CR LF, LF, and so on) is counted as one byte, regardless of the number of bytes occupied in the buffer, and the null terminator is not counted.

This interacts with the format rectangle and text limits, and a return of less than the full count can be the result. If so, a notification message is sent.

MLM_INSERT - Default Processing

The default window procedure takes no action on this message, other than to set *ulCount* to 0.

MLM_INSERT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MLM_LINEFROMCHAR

MLM_LINEFROMCHAR Field - iptFirst

iptFirst ([IPT](#))
Insertion point of interest.

MLM_LINEFROMCHAR Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_LINEFROMCHAR Return Value - ILineNum

ILineNum ([LONG](#))
Line number of insertion point.

MLM_LINEFROMCHAR - Parameters

iptFirst ([IPT](#))

Insertion point of interest.

ulReserved ([ULONG](#))

Reserved value, should be 0.

ILineNum ([LONG](#))

Line number of insertion point.

MLM_LINEFROMCHAR - Syntax

This message returns the line number corresponding to a given insertion point.

```
param1
    IPT    iptFirst    /* Insertion point of interest. */

param2
    ULONG  ulReserved /* Reserved value, should be 0. */
```

MLM_LINEFROMCHAR - Remarks

For any insertion point, the corresponding line number is returned. If the insertion point is -1, the number of the line containing the first insertion point of the selection is returned.

The term line means a line on the display after the application of word-wrap. It does not mean a line as defined by the CR LF line-break sequence.

MLM_LINEFROMCHAR - Default Processing

The default window procedure takes no action on this message, other than to set *ILineNum* to 0.

MLM_LINEFROMCHAR - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

MLM_PASTE

MLM_PASTE Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MLM_PASTE Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MLM_PASTE Return Value - ulCopy

ulCopy (ULONG)
Number of bytes transferred, counted in. CF_TEXT format.

MLM_PASTE - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

ulCopy (ULONG)
Number of bytes transferred, counted in. CF_TEXT format.

MLM_PASTE - Syntax

This message replaces the text that forms the current selection, with text from the clipboard.

```
param1
    ULONG   ulReserved /* Reserved value, should be 0. */

param2
    ULONG   ulReserved /* Reserved value, should be 0. */
```

MLM_PASTE - Remarks

The multi-line entry field control window procedure responds to this message by replacing the selected text with text from the clipboard. The text is translated from standard clipboard format, which is the same as importing with MLE_CFTTEXT format.

The text is assumed to be in the clipboard as a single contiguous data segment. This restricts the amount to the maximum segment size (64Kb).

This can cause an overflow, see MLN_OVERFLOW.

MLM_PASTE - Default Processing

The default window procedure takes no action on this message, other than to set *ulCopy* to 0.

MLM_PASTE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

MLM_QUERYBACKCOLOR

MLM_QUERYBACKCOLOR Field - IBgrndColor

IBgrndColor ([LONG](#))

Background color flag.

This flag can be one of the following:

MLE_INDEX

Indexed (solid) colors.

For version 3, or lower, of the OS/2 operating system the default is MLE_INDEX=0; MLE does not support dithered colors.

MLE_RGB

Dithered colors.

For versions, higher than version 3, of the OS/2 operating system colors default to the RGB mode.

MLM_QUERYBACKCOLOR Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

MLM_QUERYBACKCOLOR Return Value - IColor

IColor ([LONG](#))

Background color.

MLM_QUERYBACKCOLOR - Parameters

IBgrndColor ([LONG](#))

Background color flag.

This flag can be one of the following:

MLE_INDEX

Indexed (solid) colors.

For version 3, or lower, of the OS/2 operating system the default is MLE_INDEX=0; MLE does not support dithered colors.

MLE_RGB

Dithered colors.

For versions, higher than version 3, of the OS/2 operating system colors default to the RGB mode.

ulReserved ([ULONG](#))

Reserved value, should be 0.

IColor ([LONG](#))

Background color.

MLM_QUERYBACKCOLOR - Syntax

This message queries the background color.


```
param1
    LONG    lBgrndColor /* Background color flag. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */
```

MLM_QUERYBACKCOLOR - Remarks

This message returns the color in which the background is to be drawn.

The color values are the same as those used by GpiSetColor.

MLM_QUERYBACKCOLOR - Default Processing

The default window procedure takes no action on this message, other than to set *lColor* to 0.

MLM_QUERYBACKCOLOR - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

MLM_QUERYCHANGED

MLM_QUERYCHANGED Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MLM_QUERYCHANGED Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MLM_QUERYCHANGED Return Value - rc

rc (BOOL)
Current changed status.

TRUE	Text has changed since the last time that the change flag was cleared.
FALSE	Text has not changed since the last time that the change flag was cleared.

MLM_QUERYCHANGED - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

rc (BOOL)
Current changed status.

TRUE	Text has changed since the last time that the change flag was cleared.
FALSE	Text has not changed since the last time that the change flag was cleared.

MLM_QUERYCHANGED - Syntax

This message queries the changed flag.

```
param1
    ULONG  ulReserved /* Reserved value, should be 0. */

param2
    ULONG  ulReserved /* Reserved value, should be 0. */
```

MLM_QUERYCHANGED - Remarks

The multi-line entry field control window procedure responds to this message by returning the changed flag for the text without altering it. See also MLN_CHANGE.

MLM_QUERYCHANGED - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to 0 (FALSE).

MLM_QUERYCHANGED - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)
-

MLM_QUERYFIRSTCHAR

MLM_QUERYFIRSTCHAR Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_QUERYFIRSTCHAR Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_QUERYFIRSTCHAR Return Value - iptFVC

iptFVC (IPT)
First visible character.

MLM_QUERYFIRSTCHAR - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

iptFVC (IPT)
First visible character.

MLM_QUERYFIRSTCHAR - Syntax

This message queries the first visible character.

```
param1
    ULONG  ulReserved  /* Reserved value, should be 0. */

param2
    ULONG  ulReserved  /* Reserved value, should be 0. */
```

MLM_QUERYFIRSTCHAR - Remarks

Returns the insertion point immediately preceding the character visible in the upper left-hand corner of the screen. If a partial character is displayed, that character counts as the first visible character.

Note: In situations where no character is visible, because the text is scrolled to the right beyond the end of the top line, this returns the insertion point of the last character on the line (EOL not considered). In situations where there are no characters on the line, the insertion point at the beginning is returned.

MLM_QUERYFIRSTCHAR - Default Processing

The default window procedure takes no action on this message, other than to set *iptFVC* to 0.

MLM_QUERYFIRSTCHAR - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

MLM_QUERYFONT

MLM_QUERYFONT Field - pFattrs

pFattrs ([PFATTRS](#))
Font attribute structure.

MLM_QUERYFONT Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_QUERYFONT Return Value - rc

rc ([BOOL](#))
System font indicator.

TRUE	The system font is in use.
FALSE	The system font is not in use.

MLM_QUERYFONT - Parameters

pFattrs ([PFATTRS](#))
Font attribute structure.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
System font indicator.

TRUE
The system font is in use.

FALSE
The system font is not in use.

MLM_QUERYFONT - Syntax

This message queries which font is in use.

```
param1
    PFATTRS  pFattrs    /* Font attribute structure. */
param2
    ULONG    ulReserved /* Reserved value, should be 0. */
```

MLM_QUERYFONT - Remarks

This message puts the attributes of the current drawing font into the font attribute structure.

MLM_QUERYFONT - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

MLM_QUERYFONT - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

MLM_QUERYFORMATLINELENGTH

MLM_QUERYFORMATLINELENGTH Field - iptStart

iptStart (IPT)
Insertion point to count from.

MLM_QUERYFORMATLINELENGTH Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MLM_QUERYFORMATLINELENGTH Return Value - iptLine

iptLine (IPT)
Count of bytes to end of line.

MLM_QUERYFORMATLINELENGTH - Parameters

iptStart (IPT)
Insertion point to count from.

ulReserved (ULONG)
Reserved value, should be 0.

iptLine (IPT)
Count of bytes to end of line.

MLM_QUERYFORMATLINELENGTH - Syntax

This message returns the number of bytes to end of line after formatting has been applied.

```
param1
    IPT    iptStart    /* Insertion point to count from. */
```

```
param2
    ULONG    ulReserved    /* Reserved value, should be 0. */
```

MLM_QUERYFORMATLINELENGTH - Remarks

For any insertion point, the number of bytes between that insertion point and the end of the line is returned, after the current formatting is applied. If the insertion point is -1, the cursor position is used. This message differs from [MLM_QUERYLINELENGTH](#) in that the byte count returned reflects the effects of the current formatting set by [MLM_FORMAT](#).

MLM_QUERYFORMATLINELENGTH - Default Processing

The default window procedure takes no action on this message, other than to set *iptLine* to 0.

MLM_QUERYFORMATLINELENGTH - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

MLM_QUERYFORMATRECT

MLM_QUERYFORMATRECT Field - pFormatRect

pFormatRect ([PPOINTL](#))
Format dimensions.

The size of the current limiting dimensions.

MLM_QUERYFORMATRECT Field - flFlags

flFlags ([ULONG](#))

Flags governing interpretation of dimensions.

An array of MLFFMTRECT_* flags defined under the *flFlags* field of the [MLM_SETFORMATRECT](#) message.

MLM_QUERYFORMATRECT Return Value - ulReserved

ulReserved ([ULONG](#))

Reserved value.

MLM_QUERYFORMATRECT - Parameters

pFormatRect ([PPOINTL](#))

Format dimensions.

The size of the current limiting dimensions.

flFlags ([ULONG](#))

Flags governing interpretation of dimensions.

An array of MLFFMTRECT_* flags defined under the *flFlags* field of the [MLM_SETFORMATRECT](#) message.

ulReserved ([ULONG](#))

Reserved value.

MLM_QUERYFORMATRECT - Syntax

This message queries the format dimensions and mode.

```
param1
    PPOINTL    pFormatRect    /* Format dimensions. */

param2
    ULONG      flFlags          /* Flags governing interpretation of dimensions. */
```

MLM_QUERYFORMATRECT - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

MLM_QUERYFORMATRECT - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Glossary](#)

MLM_QUERYFORMATTEXTLENGTH

MLM_QUERYFORMATTEXTLENGTH Field - iptStart

iptStart ([IPT](#))
Insertion point to start from.

MLM_QUERYFORMATTEXTLENGTH Field - ulScan

ulScan ([ULONG](#))
Number of characters to convert to bytes.

0xFFFFFFFF	Convert until end of line
other	Convert specified number of characters.

MLM_QUERYFORMATTEXTLENGTH Return Value - ulText

ulText ([ULONG](#))
Count of bytes in text after formatting.

MLM_QUERYFORMATTEXTLENGTH - Parameters

iptStart ([IPT](#))
Insertion point to start from.

ulScan ([ULONG](#))
Number of characters to convert to bytes.

0xFFFFFFFF	Convert until end of line
other	Convert specified number of characters.

ulText ([ULONG](#))
Count of bytes in text after formatting.

MLM_QUERYFORMATTEXTLENGTH - Syntax

This message returns the length of a specified range of characters after the current formatting has been applied.

```
param1
    IPT      iptStart /* Insertion point to start from. */

param2
    ULONG    ulScan   /* Number of characters to convert to bytes. */
```

MLM_QUERYFORMATTEXTLENGTH - Remarks

This message returns the length in bytes of a range of characters after the current formatting is applied. This differs from [MLM_QUERYTEXTLENGTH](#) in that:

- A range of insertion points can be queried.
- The byte count returned reflects the effects of the current formatting set by [MLM_FORMAT](#).

MLM_QUERYFORMATTEXTLENGTH - Default Processing

The default window procedure takes no action on this message, other than to set *ulText* to 0.

MLM_QUERYFORMATTEXTLENGTH - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)

MLM_QUERYIMPORTEXP

MLM_QUERYIMPORTEXP Field - Buff

Buff ([PVOID *](#))
Transfer buffer.

MLM_QUERYIMPORTEXP Field - pulLength

pulLength ([PULONG](#))
Size of transfer buffer in bytes.

MLM_QUERYIMPORTEXP Return Value - rc

rc ([ULONG](#))
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

MLM_QUERYIMPORTEXP - Parameters

Buff ([PVOID *](#))
Transfer buffer.

pulLength ([PULONG](#))
Size of transfer buffer in bytes.

rc ([ULONG](#))
Success indicator.

TRUE

FALSE	Successful completion.
	Error occurred.

MLM_QUERYIMPORTEXPOR - Syntax

This message queries the current transfer buffer.

```
param1
    PVOID * Buff          /* Transfer buffer. */

param2
    PULONG pulLength      /* Size of transfer buffer in bytes. */
```

MLM_QUERYIMPORTEXPOR - Remarks

This message returns the values from the most recent [MLM_SETIMPORTEXPOR](#), or 0 for either value if it has not been set.

MLM_QUERYIMPORTEXPOR - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to 0 (FALSE).

MLM_QUERYIMPORTEXPOR - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

MLM_QUERYLINECOUNT

MLM_QUERYLINECOUNT Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MLM_QUERYLINECOUNT Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MLM_QUERYLINECOUNT Return Value - ulLines

ulLines (ULONG)
The number of lines of text.

MLM_QUERYLINECOUNT - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

ulLines (ULONG)
The number of lines of text.

MLM_QUERYLINECOUNT - Syntax

This message queries the number of lines of text.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */
param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

MLM_QUERYLINECOUNT - Remarks

The term line means a line on the display after the application of word-wrap. It does not mean a line as defined by the CR LF line-break sequence.

The multi-line edit control always maintains one CR LF line-break in the buffer, therefore the number of lines returned may be one greater than the number actually visible.

MLM_QUERYLINECOUNT - Default Processing

The default window procedure takes no action on this message, other than to set *ulLines* to 0.

MLM_QUERYLINECOUNT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MLM_QUERYLINELENGTH

MLM_QUERYLINELENGTH Field - iptStart

iptStart ([IPT](#))
Insertion point to count from.

MLM_QUERYLINELENGTH Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_QUERYLINELENGTH Return Value - iptLine

iptLine (IPT)
Count of bytes to end of line.

MLM_QUERYLINELENGTH - Parameters

iptStart (IPT)
Insertion point to count from.

ulReserved (ULONG)
Reserved value, should be 0.

iptLine (IPT)
Count of bytes to end of line.

MLM_QUERYLINELENGTH - Syntax

This message returns the number of bytes between a given insertion point and the end of line.

```
param1
    IPT    iptStart    /* Insertion point to count from. */
param2
    ULONG  ulReserved /* Reserved value, should be 0. */
```

MLM_QUERYLINELENGTH - Remarks

For any insertion point, the number of bytes between that insertion point and the end of the line is returned. If the insertion point is -1, the cursor position is used. If the line contains a hard line-break, it is counted as one byte.

The term line means a line on the display after the application of word-wrap. It does not mean a line as defined by the CR LF line-break sequence, although a display line might be terminated by such a sequence.

MLM_QUERYLINELENGTH - Default Processing

The default window procedure takes no action on this message, other than to set *iptLine* to 0.

MLM_QUERYLINELENGTH - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

MLM_QUERYREADONLY

MLM_QUERYREADONLY Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_QUERYREADONLY Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_QUERYREADONLY Return Value - rc

rc ([BOOL](#))
Current read-only status.

TRUE	Read-only mode is set.
FALSE	Read-only mode is cleared.

MLM_QUERYREADONLY - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Current read-only status.

TRUE	Read-only mode is set.
FALSE	Read-only mode is cleared.

MLM_QUERYREADONLY - Syntax

This message queries the read-only mode.

```
param1
    ULONG  ulReserved  /*  Reserved value, should be 0.  */
param2
    ULONG  ulReserved  /*  Reserved value, should be 0.  */
```

MLM_QUERYREADONLY - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

MLM_QUERYREADONLY - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Default Processing](#)
 - [Glossary](#)

MLM_QUERYSEL

MLM_QUERYSEL Field - usQueryMode

usQueryMode (USHORT)

Query Mode.

MLFQS_MINMAXSEL

Return both minimum and maximum points of selection in a format compatible with the [EM_QUERYSEL](#) message.

MLFQS_MINSEL

Return minimum insertion point of selection.

MLFQS_MAXSEL

Return maximum insertion point of selection.

MLFQS_ANCHORSEL

Return anchor point of selection.

MLFQS_CURSORSEL

Return cursor point of selection.

MLM_QUERYSEL Field - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

MLM_QUERYSEL Field - sMinSel

sMinSel (SHORT)

Minimum insertion point of selection.

This value is rounded down to 65 535, if necessary.

ReturnCode contains *sMinSel* and *sMaxSel* for a *usQueryMode* of MLFQS_MINMAXSEL.

MLM_QUERYSEL Field - sMaxSel

sMaxSel (SHORT)

Maximum insertion point of selection.

This value is rounded down to 65 535 if necessary.

ReturnCode contains *sMinSel* and *sMaxSel* for a *usQueryMode* of MLFQS_MINMAXSEL.

MLM_QUERYSEL Field - ipt

ipt ([IPT](#))

Requested insertion point.

ReturnCode contains *ipt* for a *usQueryMode* of MLFQS_MINSEL, MLFQS_MAXSEL, MLFQS_ANCHORSEL, or MLFQS_CURSORSEL.

MLM_QUERYSEL - Parameters

usQueryMode ([USHORT](#))

Query Mode.

MLFQS_MINMAXSEL

Return both minimum and maximum points of selection in a format compatible with the [EM_QUERYSEL](#) message.

MLFQS_MINSEL

Return minimum insertion point of selection.

MLFQS_MAXSEL

Return maximum insertion point of selection.

MLFQS_ANCHORSEL

Return anchor point of selection.

MLFQS_CURSORSEL

Return cursor point of selection.

ulReserved ([ULONG](#))

Reserved value, should be 0.

sMinSel ([SHORT](#))

Minimum insertion point of selection.

This value is rounded down to 65 535, if necessary.

ReturnCode contains *sMinSel* and *sMaxSel* for a *usQueryMode* of MLFQS_MINMAXSEL.

sMaxSel ([SHORT](#))

Maximum insertion point of selection.

This value is rounded down to 65 535 if necessary.

ReturnCode contains *sMinSel* and *sMaxSel* for a *usQueryMode* of MLFQS_MINMAXSEL.

ipt ([IPT](#))

Requested insertion point.

ReturnCode contains *ipt* for a *usQueryMode* of MLFQS_MINSEL, MLFQS_MAXSEL, MLFQS_ANCHORSEL, or MLFQS_CURSORSEL.

MLM_QUERYSEL - Syntax

This message returns the location of the selection.

param1

[USHORT](#) *usQueryMode* /* Query Mode. */

```

param2
    ULONG    ulReserved    /* Reserved value, should be 0. */

returns
    SHORT    sMinSel       /* Minimum insertion point of selection. */
    SHORT    sMaxSel       /* Maximum insertion point of selection. */
    IPT      ipt           /* Requested insertion point. */

```

MLM_QUERYSEL - Remarks

This message returns the location of the selection in several different forms. The insertion points lie between characters, and start at a zero origin before the first character in the MLE. Subtracting the minimum from the maximum gives the number of characters in the selection. *This is not necessarily the number of bytes of ASCII.* The line-break character is a CR LF (2 bytes) and all DBCS characters are 2 bytes. To determine the number of bytes, use [MLM_QUERYFORMATTEXTLENGTH](#), being sure that the format choice set by [MLM_FORMAT](#) is set to what is used when the data is exported from the MLE (for example, MLE_CFTTEXT for [MLM_QUERYSELTEXT](#)).

Note the following:

- If anchor point > cursor point, minimum point = cursor point and maximum point = anchor point.
- If anchor point < cursor point, minimum point = anchor point and maximum point = cursor point.

MLM_QUERYSEL - Default Processing

The default window procedure takes no action on this message, other than to set *ReturnCode* to 0.

MLM_QUERYSEL - Examples

This example sends two MLM_QUERYSEL messages to obtain the beginning and ending points of the current selection, sends an MLM_SETIMPORTEXP message to set up the export buffer, and then sends an MLM_EXPORT message to export the selection into the buffer.

```

LONG lStart, cch;
CHAR szBuf[500];

lStart = (LONG) WinSendMsg(hwndMle, MLM_QUERYSEL,
    (MPARAM) MLFQS_MINSEL, (MPARAM) 0L);
cch = lStart - (LONG) WinSendMsg(hwndMle, MLM_QUERYSEL,
    (MPARAM) MLFQS_MAXSEL, (MPARAM) 0L);
WinSendMsg(hwndMle, MLM_SETIMPORTEXP,
    (MPARAM) szBuf, (MPARAM) sizeof(szBuf));
WinSendMsg(hwndMle, MLM_EXPORT, (MPARAM) &lStart, (MPARAM) &cch);

```

MLM_QUERYSEL - Topics

Select an item:

[Syntax](#)
[Parameters](#)

MLM_QUERYSELTEXT

MLM_QUERYSELTEXT Field - pchBuff

pchBuff ([PCHAR](#))
Character buffer for text string.

MLM_QUERYSELTEXT Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_QUERYSELTEXT Return Value - ulCount

ulCount ([ULONG](#))
Number of bytes to put into text string.

MLM_QUERYSELTEXT - Parameters

pchBuff ([PCHAR](#))
Character buffer for text string.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulCount ([ULONG](#))
Number of bytes to put into text string.

MLM_QUERYSELTEXT - Syntax

This message copies the currently selected text into a buffer.

```
param1
    PCHAR  pchBuff    /* Character buffer for text string. */

param2
    ULONG  ulReserved /* Reserved value, should be 0. */
```

MLM_QUERYSELTEXT - Remarks

This message copies the currently selected text into the buffer pointed to by *pchBuff*. The text string is null-terminated. The byte count includes the text in CF_TEXT format (CR LF) and the null terminator.

MLM_QUERYSELTEXT - Default Processing

The default window procedure takes no action on this message, other than to set *ulCount* to 0.

MLM_QUERYSELTEXT - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

MLM_QUERYTABSTOP

MLM_QUERYTABSTOP Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

MLM_QUERYTABSTOP Field - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

MLM_QUERYTABSTOP Return Value - pixTabset

pixTabset (PIX)

Tab width in pels.

< 0

An error occurred.

Other

The pel interval at which tab stops are placed.

MLM_QUERYTABSTOP - Parameters

ulReserved (ULONG)

Reserved value, should be 0.

ulReserved (ULONG)

Reserved value, should be 0.

pixTabset (PIX)

Tab width in pels.

< 0

An error occurred.

Other

The pel interval at which tab stops are placed.

MLM_QUERYTABSTOP - Syntax

This message queries the pel interval at which tab stops are placed.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

MLM_QUERYTABSTOP - Remarks

This message fails and returns a negative value, if the reserved values are not 0.

MLM_QUERYTABSTOP - Default Processing

The default window procedure takes no action on this message, other than to set *pixTabset* to 0.

MLM_QUERYTABSTOP - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)
-

MLM_QUERYTEXTCOLOR

MLM_QUERYTEXTCOLOR Field - ITextColor

ITextColor ([LONG](#))
Text color flag.

This flag can be one of the following:

MLE_INDEX	Indexed (solid) colors. For version 3, or lower, of the OS/2 operating system the default is MLE_INDEX=0; MLE does not support dithered colors.
MLE_RGB	Dithered colors. For versions, higher than version 3, of the OS/2 operating system colors default to the RGB mode.

MLM_QUERYTEXTCOLOR Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MLM_QUERYTEXTCOLOR Return Value - IColor

IColor (LONG)
Text color.

MLM_QUERYTEXTCOLOR - Parameters

ITextColor (LONG)
Text color flag.

This flag can be one of the following:

MLE_INDEX	Indexed (solid) colors.
	For version 3, or lower, of the OS/2 operating system the default is MLE_INDEX=0; MLE does not support dithered colors.
MLE_RGB	Dithered colors.
	For versions, higher than version 3, of the OS/2 operating system colors default to the RGB mode.

ulReserved (ULONG)
Reserved value, should be 0.

IColor (LONG)
Text color.

MLM_QUERYTEXTCOLOR - Syntax

This message queries the text color.

```
param1
    LONG    lTextColor /* Text color flag. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */
```

MLM_QUERYTEXTCOLOR - Remarks

This message returns the color in which text is to be drawn.

The color values are the same as those used by GpiSetColor.

MLM_QUERYTEXTCOLOR - Default Processing

The default window procedure takes no action on this message, other than to set */Color* to 0.

MLM_QUERYTEXTCOLOR - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

MLM_QUERYTEXTLENGTH

MLM_QUERYTEXTLENGTH Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

MLM_QUERYTEXTLENGTH Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

MLM_QUERYTEXTLENGTH Return Value - iptText

iptText ([IPT](#))
Count of text in bytes.

MLM_QUERYTEXTLENGTH - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

iptText ([IPT](#))
Count of text in bytes.

MLM_QUERYTEXTLENGTH - Syntax

This message returns the number of characters in the text.

```
param1
    ULONG    ulReserved /* Reserved value, should be 0. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */
```

MLM_QUERYTEXTLENGTH - Remarks

This message returns the number of characters in the text. Hard line-breaks are counted as 1 and soft line-breaks as 0.

This message differs from the [WinQueryWindowTextLength](#) call in that it returns a [LONG](#).

MLM_QUERYTEXTLENGTH - Default Processing

The default window procedure takes no action on this message, other than to set *iptText* to 0.

MLM_QUERYTEXTLENGTH - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

MLM_QUERYTEXTLIMIT

MLM_QUERYTEXTLIMIT Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_QUERYTEXTLIMIT Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_QUERYTEXTLIMIT Return Value - ISize

ISize ([LONG](#))
Maximum number of bytes allowed in the MLE.

MLM_QUERYTEXTLIMIT - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ISize ([LONG](#))

Maximum number of bytes allowed in the MLE.

MLM_QUERYTEXTLIMIT - Syntax

This message queries the maximum number of bytes that a multi-line entry field control can contain.

```
param1
    ULONG    ulReserved /* Reserved value, should be 0. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */
```

MLM_QUERYTEXTLIMIT - Remarks

The multi-line entry field control window procedure responds to this message by returning the current limit set, either by default, or by [MLM_SETTEXTLIMIT](#). If the limit is unbounded, a non-positive value is returned.

MLM_QUERYTEXTLIMIT - Default Processing

The default window procedure takes no action on this message, other than to set *Size* to 0.

MLM_QUERYTEXTLIMIT - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

MLM_QUERYUNDO

MLM_QUERYUNDO Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MLM_QUERYUNDO Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MLM_QUERYUNDO Field - usOperation

usOperation (USHORT)
Operation that can be undone or redone.

0	An undo or redo operation is not possible.
WM_CHAR	A WM_CHAR message, or messages for a simple string of keystrokes, can be undone or redone.
MLM_SETFONT	A MLM_SETFONT message can be undone or redone.
MLM_SETTEXTCOLOR	A MLM_SETTEXTCOLOR message can be undone or redone for both background and foreground color.
MLM_CUT	A MLM_CUT message can be undone or redone.
MLM_PASTE	A MLM_PASTE message can be undone or redone.
MLM_CLEAR	A MLM_CLEAR message can be undone or redone.

MLM_QUERYUNDO Field - rc

rc (BOOL)
Undo or redo indicator.

TRUE	An undo is possible.
FALSE	A redo is possible.

MLM_QUERYUNDO - Parameters

ulReserved (ULONG)	Reserved value, should be 0.
ulReserved (ULONG)	Reserved value, should be 0.
usOperation (USHORT)	Operation that can be undone or redone.
0	An undo or redo operation is not possible.
WM_CHAR	A WM_CHAR message, or messages for a simple string of keystrokes, can be undone or redone.
MLM_SETFONT	A MLM_SETFONT message can be undone or redone.
MLM_SETTEXTCOLOR	A MLM_SETTEXTCOLOR message can be undone or redone for both background and foreground color.
MLM_CUT	A MLM_CUT message can be undone or redone.
MLM_PASTE	A MLM_PASTE message can be undone or redone.
MLM_CLEAR	A MLM_CLEAR message can be undone or redone.
rc (BOOL)	Undo or redo indicator.
TRUE	An undo is possible.
FALSE	A redo is possible.

MLM_QUERYUNDO - Syntax

This message queries the undo or redo operations that are possible.

```

param1
    ULONG    ulReserved    /* Reserved value, should be 0. */

param2
    ULONG    ulReserved    /* Reserved value, should be 0. */

returns
    USHORT   usOperation    /* Operation that can be undone or redone. */
    BOOL     rc              /* Undo or redo indicator. */

```

MLM_QUERYUNDO - Default Processing

The default window procedure takes no action on this message, other than to set *reply* to 0.

MLM_QUERYUNDO - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Glossary](#)

MLM_QUERYWRAP

MLM_QUERYWRAP Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_QUERYWRAP Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_QUERYWRAP Return Value - rc

rc ([BOOL](#))
Wrap flag.

TRUE	Word-wrap enabled
FALSE	Word-wrap disabled.

MLM_QUERYWRAP - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Wrap flag.

TRUE	Word-wrap enabled
FALSE	Word-wrap disabled.

MLM_QUERYWRAP - Syntax

This message queries the wrap flag.

```
param1
    ULONG  ulReserved  /* Reserved value, should be 0. */

param2
    ULONG  ulReserved  /* Reserved value, should be 0. */
```

MLM_QUERYWRAP - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

MLM_QUERYWRAP - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Default Processing](#)
 - [Glossary](#)

MLM_RESETUNDO

MLM_RESETUNDO Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MLM_RESETUNDO Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MLM_RESETUNDO Field - usOperation

usOperation (USHORT)
Operation that can be undone or redone.

- 0
An undo or redo operation is not possible.
- WM_CHAR
A WM_CHAR message, or messages for a simple string of keystrokes, can be undone or redone.
- MLM_SETFONT
A MLM_SETFONT message can be undone or redone.
- MLM_SETTEXTCOLOR
A MLM_SETTEXTCOLOR message can be undone or redone for both background and foreground color.
- MLM_CUT
A MLM_CUT message can be undone or redone.
- MLM_PASTE
A MLM_PASTE message can be undone or redone.
- MLM_CLEAR
A MLM_CLEAR message can be undone or redone.

MLM_RESETUNDO Field - rc

rc (BOOL)
Undo or redo indicator.

- TRUE
An undo is possible.
- FALSE
A redo is possible.

MLM_RESETUNDO - Parameters

ulReserved (ULONG)

Reserved value, should be 0.

ulReserved (ULONG)

Reserved value, should be 0.

usOperation (USHORT)

Operation that can be undone or redone.

0

An undo or redo operation is not possible.

WM_CHAR

A [WM_CHAR](#) message, or messages for a simple string of keystrokes, can be undone or redone.

MLM_SETFONT

A [MLM_SETFONT](#) message can be undone or redone.

MLM_SETTEXTCOLOR

A [MLM_SETTEXTCOLOR](#) message can be undone or redone for both background and foreground color.

MLM_CUT

A [MLM_CUT](#) message can be undone or redone.

MLM_PASTE

A [MLM_PASTE](#) message can be undone or redone.

MLM_CLEAR

A [MLM_CLEAR](#) message can be undone or redone.

rc (BOOL)

Undo or redo indicator.

TRUE

An undo is possible.

FALSE

A redo is possible.

MLM_RESETUNDO - Syntax

This message resets the undo state to indicate that no undo operations are possible.

```
param1
    ULONG    ulReserved    /* Reserved value, should be 0. */

param2
    ULONG    ulReserved    /* Reserved value, should be 0. */

returns
    USHORT   usOperation    /* Operation that can be undone or redone. */
    BOOL     rc              /* Undo or redo indicator. */
```

MLM_RESETUNDO - Remarks

This message resets the undo state of the MLE to indicate that the last operation cannot be undone (null return from [MLM_QUERYUNDO](#)). This can be used by the application when it performs an operation that it can undo, that supersedes the last MLE operation. The application can then reset its own undo state upon receipt of an MLN_CHANGE, indicating that later changes have occurred through the MLE.

MLM_RESETUNDO - Default Processing

The default window procedure takes no action on this message, other than to set *ReturnCode* to 0.

MLM_RESETUNDO - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

MLM_SEARCH

MLM_SEARCH Field - ulStyle

ulStyle ([ULONG](#))

Style flags.

MLFSEARCH_CASESENSITIVE

If set, only exact matches are considered a successful match. If not set, any case-combination of the correct characters in the correct sequence is considered a successful match.

MLFSEARCH_SELECTMATCH

If set, the MLE selects the text and scrolls it into view when found, just as if the application had sent an [MLM_SETSEL](#) message. This is not done if MLFSEARCH_CHANGEALL is also indicated.

MLFSEARCH_CHANGEALL

Using the [MLE_SEARCHDATA](#) structure specified in *pse*, all occurrences of *pchFind* are found, searching from *iptStart* to *iptStop*, and replacing them with *pchReplace*. If this style is selected, the *cchFound* field has no meaning, and the *iptStart* value points to the place where the search stopped, or is the same as *iptStop* because the search has not been stopped at any of the found strings. The current cursor location is not moved. However, any existing selection is deselected.

MLM_SEARCH Field - pse

pse ([PMLE_SEARCHDATA](#))
Search specification structure.

MLM_SEARCH Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	The search was successful.
FALSE	The search was unsuccessful.

MLM_SEARCH - Parameters

ulStyle ([ULONG](#))
Style flags.

MLFSEARCH_CASESENSITIVE
If set, only exact matches are considered a successful match. If not set, any case-combination of the correct characters in the correct sequence is considered a successful match.

MLFSEARCH_SELECTMATCH
If set, the MLE selects the text and scrolls it into view when found, just as if the application had sent an [MLM_SETSEL](#) message. This is not done if MLFSEARCH_CHANGEALL is also indicated.

MLFSEARCH_CHANGEALL
Using the [MLE_SEARCHDATA](#) structure specified in *pse*, all occurrences of *pchFind* are found, searching from *iptStart* to *iptStop*, and replacing them with *pchReplace*. If this style is selected, the *cchFound* field has no meaning, and the *iptStart* value points to the place where the search stopped, or is the same as *iptStop* because the search has not been stopped at any of the found strings. The current cursor location is not moved. However, any existing selection is deselected.

pse ([PMLE_SEARCHDATA](#))
Search specification structure.

rc ([BOOL](#))
Success indicator.

TRUE	The search was successful.
FALSE	The search was unsuccessful.

MLM_SEARCH - Syntax

This message searches for a specified text string.

```
param1
    ULONG          ulStyle /* Style flags. */

param2
    PMLE_SEARCHDATA pse /* Search specification structure. */
```

MLM_SEARCH - Remarks

This message searches the MLE text for a specified string, starting at a specified insertion point and continuing until the second specified insertion point has been reached, or the requested string has been matched.

When an MLM_SEARCH message is sent, the text is scanned starting with the character that follows the insertion point indicated in the *ipStart* field of the [MLE_SEARCHDATA](#) structure. The search proceeds until the point indicated in the *ipStop* field, until a match is found, or until TRUE is returned from MLN_SEARCHPAUSE notification (see [WM_CONTROL \(in Multiline Entry Fields\)](#)). If a negative value is specified for the *ipStart*, the current cursor point is used. If a negative value is specified for *ipStop*, the end of the text is used. If *ipStop*, is less than or equal to *ipStart*, after performing the two indicated substitutions, the search wraps from the end of the text to the beginning of the text.

If the MLFSEARCH_CASESENSITIVE option is specified, the bytes of the search string must exactly match those in the text. If MLFSEARCH_CASESENSITIVE is not specified, the [WinUpperChar](#) of the search string must match the [WinUpperChar](#) of the text.

When a match is found, the *ipStart* field of the search specification structure is set to indicate the insertion point immediately preceding the first character of the match, and the *cchFind* field is set to indicate the number of characters in the match. The cursor selection is not altered unless MLFSEARCH_SELECTMATCH is specified. If it is, an [MLM_SETSEL](#) is done with the anchor point at *ipStart* and the cursor at *ipStart + cchFind*.

While searching, the MLE occasionally sends an MLN_SEARCHPAUSE notification message. If the owner responds to this message with the value TRUE, the MLE stops the search. When a search is stopped from MLN_SEARCHPAUSE, *ipStart* is set to the point where the search terminated. If the response is FALSE, the search continues (see also the definition of MLN_SEARCHPAUSE). The interval at which MLN_SEARCHPAUSE notifications are sent is implementation-dependent, but must not exceed reasonable user-response thresholds, nor should it be so often as to introduce undue messaging overhead. Sending this notification every half second is a reasonable compromise.

When no match is found the *ipStart* value is unchanged.

If the application needs to continue the search, the proper way is to change the *ipStart* value to be the point following the string found, adjusting for any text changes done after the search that may have moved the relative location of the point.

Applications using this message are advised to change the system pointer to the wait icon (clock face) if it is expected that the search will take some time.

MLM_SEARCH - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

MLM_SEARCH - Examples

This example searches for all occurrences of the word "Bonnie" and replaces them with the word "Jeannette."

```
MLE_SEARCHDATA search;
```

```

search.cb = sizeof(search);
search.pchFind = "bonnie";
search.pchReplace = "jeannette";
search.cchFind = 6;
search.cchReplace = 9;
search.iptStart = 0; /* from the beginning of the text */
search.iptStop = -1; /* to the end of the text */
WinSendMsg(hwndMle, MLM_SEARCH, MLFSEARCH_CHANGEALL, (MPARAM) &search)

```

MLM_SEARCH - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Examples](#)
[Glossary](#)

MLM_SETBACKCOLOR

MLM_SETBACKCOLOR Field - IColor

IColor ([LONG](#))
Color.

MLM_SETBACKCOLOR Field - IBgrndColor

IBgrndColor ([LONG](#))
Background color flag.

This flag can be set to one of the following:

MLE_INDEX	Indexed (solid) colors.
-----------	-------------------------

For version 3, or lower, of the OS/2 operating system the default is MLE_INDEX=0; MLE does not support dithered colors.

MLE_RGB	Dithered colors.
---------	------------------

For versions, higher than version 3, of the OS/2 operating system colors default to the RGB mode.

MLM_SETBACKCOLOR Return Value - IOldColor

IOldColor ([LONG](#))
Color previously used.

MLM_SETBACKCOLOR - Parameters

IColor ([LONG](#))
Color.

IBgrndColor ([LONG](#))
Background color flag.

This flag can be set to one of the following:

MLE_INDEX Indexed (solid) colors.

For version 3, or lower, of the OS/2 operating system the default is MLE_INDEX=0; MLE does not support dithered colors.

MLE_RGB Dithered colors.

For versions, higher than version 3, of the OS/2 operating system colors default to the RGB mode.

IOldColor ([LONG](#))
Color previously used.

MLM_SETBACKCOLOR - Syntax

This message sets the background color.

```
param1
    LONG  lColor      /* Color. */

param2
    LONG  lBgrndColor /* Background color flag. */
```

MLM_SETBACKCOLOR - Remarks

This message sets the color in which the MLE background is to be drawn, and updates the display as necessary.

The color values are the same as those used by GpiSetColor.

MLM_SETBACKCOLOR - Default Processing

The default window procedure takes no action on this message, other than to set */OldColor* to 0.

MLM_SETBACKCOLOR - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MLM_SETCHANGED

MLM_SETCHANGED Field - usChangedNew

usChangedNew ([USHORT](#))
Value to set changed flag to.

TRUE	Changed flag set.
FALSE	Changed flag cleared.

MLM_SETCHANGED Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_SETCHANGED Return Value - rc

rc (BOOL)

Changed status before message was processed.

TRUE

Text has changed since the last time that the change flag was cleared.

FALSE

Text has not changed since the last time that the change flag was cleared.

MLM_SETCHANGED - Parameters

usChangedNew (USHORT)

Value to set changed flag to.

TRUE

Changed flag set.

FALSE

Changed flag cleared.

ulReserved (ULONG)

Reserved value, should be 0.

rc (BOOL)

Changed status before message was processed.

TRUE

Text has changed since the last time that the change flag was cleared.

FALSE

Text has not changed since the last time that the change flag was cleared.

MLM_SETCHANGED - Syntax

This message sets or clears the changed flag.

```
param1
    USHORT    usChangedNew /* Value to set changed flag to. */
param2
    ULONG     ulReserved   /* Reserved value, should be 0. */
```

MLM_SETCHANGED - Remarks

This message can generate a MLN_CHANGE notification.

MLM_SETCHANGED - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

MLM_SETCANGED - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

MLM_SETFIRSTCHAR

MLM_SETFIRSTCHAR Field - iptFVC

iptFVC ([IPT](#))
Insertion point to place in top left-hand corner.

MLM_SETFIRSTCHAR Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_SETFIRSTCHAR Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	An error occurred.

MLM_SETFIRSTCHAR - Parameters

iptFVC ([IPT](#))
Insertion point to place in top left-hand corner.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	An error occurred.

MLM_SETFIRSTCHAR - Syntax

This message sets the first visible character.

```
param1
    IPT    iptFVC    /* Insertion point to place in top left-hand corner. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

MLM_SETFIRSTCHAR - Remarks

This message scrolls the text to place the character following the insertion point into the upper left-hand corner of the window. If the insertion point specified is beyond the end of a line, or the end of the file, it is resolved in the same way as it is for a mouse click.

MLM_SETFIRSTCHAR - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

MLM_SETFIRSTCHAR - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)

MLM_SETFONT

MLM_SETFONT Field - pFattrs

pFattrs (PFATTRS)	
Font attribute structure.	
NULL	The system font is set.
other	The specified font is set.

MLM_SETFONT Field - ulReserved

ulReserved (ULONG)	
Reserved value, should be 0.	

MLM_SETFONT Return Value - rc

rc (BOOL)	
Success indicator.	
TRUE	The font was successfully set.
FALSE	An error occurred.

MLM_SETFONT - Parameters

pFattrs (PFATTRS)	
Font attribute structure.	
NULL	

	The system font is set.
other	The specified font is set.
ulReserved (ULONG)	
	Reserved value, should be 0.
rc (BOOL)	
	Success indicator.
TRUE	The font was successfully set.
FALSE	An error occurred.

MLM_SETFONT - Syntax

This message sets a font.

```
param1
    PFATTRS  pFattrs    /* Font attribute structure. */
param2
    ULONG    ulReserved /* Reserved value, should be 0. */
```

MLM_SETFONT - Remarks

For any *PFATTRS*, this message sets the display to use the appropriate font. If NULL, the system font is used. The screen is updated appropriately.

This can cause an overflow, see MLN_OVERFLOW.

When setting an outline font it is necessary to ensure that the FATTRS structure contains the correct maximum baseline extent and average character width for the desired point size and that the font use is marked as FATTR_FONTUSE_TRANSFORMABLE.

Baseline extent and character width are calculated by multiplying the desired point size by the current display device font resolution (CAPS_VERTICAL_FONT_RES and CAPS_HORIZONTAL_FONT_RES; see [DevQueryCaps](#)) and dividing by 72, the number of points in an inch.

MLM_SETFONT - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

MLM_SETFONT - Examples

This example retrieves the current font information, changes it to italic, and sets it using the MLM_SETFONT message.

```
FATTRS fat;
fat.usRecordLength = sizeof(FATTRS);
WinSendMsg(hwndMle, MLM_QUERYFONT, (MPARAM) &fat, (MPARAM) 0L);
fat.fsSelection = FATTR_SEL_ITALIC;
WinSendMsg(hwndMle, MLM_SETFONT, (MPARAM) &fat, (MPARAM) 0);
```

MLM_SETFONT - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Examples](#)
[Glossary](#)

MLM_SETFORMATRECT

MLM_SETFORMATRECT Field - pFormatRect

pFormatRect ([PPOINTL](#))

New format dimensions.

NULL

A null value sets both dimensions to the current window size.

other

The structure is a pair of [LONGs](#) designating the diagonally-opposite corner of the rectangle, assuming 0,0 for the first. Therefore, they are the width and height in pels of the format rectangle. These dimensions are used as the word-wrap and text-size limiting boundaries. Negative values for either dimension cause the MLE to substitute the current window size (the MLE window rectangle minus margins).

If the rectangle specified has either, or both, of the limits set, and the size is inadequate to contain the text, *rc* is set to FALSE and the rectangle dimensions are replaced with the overflow amounts.

MLM_SETFORMATRECT Field - flFlags

flFlags ([ULONG](#))

Flags governing interpretation of dimensions.

MLFFMTRECT_MATCHWINDOW

The dimensions of the format rectangle are always to be kept the same as the window size minus the margins. This causes the MLE implicitly to do a MLM_SETFORMATRECT each time the window is resized, and effectively causes any other dimensions to be ignored. Resizing of the window can cause this setting to be automatically negated (see MLN_OVERFLOW).

MLFFMTRECT_LIMITHORZ

The width of any line in the MLE cannot exceed the given horizontal dimension. If word-wrap is on, this limit has no effect. Word-wrap can result in trailing blanks beyond the right limit. These do not cause an overflow notification.

MLFFMTRECT_LIMITVERT

The vertical height of the total text, as displayed, is limited to that which fits totally within the vertical dimension of the format rectangle.

MLM_SETFORMATRECT Return Value - rc

rc (BOOL)

Success indicator.

TRUE

Successful completion

FALSE

An error occurred.

MLM_SETFORMATRECT - Parameters

pFormatRect (PPOINTL)

New format dimensions.

NULL

A null value sets both dimensions to the current window size.

other

The structure is a pair of LONGs designating the diagonally-opposite corner of the rectangle, assuming 0,0 for the first. Therefore, they are the width and height in pels of the format rectangle. These dimensions are used as the word-wrap and text-size limiting boundaries. Negative values for either dimension cause the MLE to substitute the current window size (the MLE window rectangle minus margins).

If the rectangle specified has either, or both, of the limits set, and the size is inadequate to contain the text, rc is set to FALSE and the rectangle dimensions are replaced with the overflow amounts.

flFlags (ULONG)

Flags governing interpretation of dimensions.

MLFFMTRECT_MATCHWINDOW

The dimensions of the format rectangle are always to be kept the same as the window size minus the margins. This causes the MLE implicitly to do a MLM_SETFORMATRECT each time the window is resized, and effectively causes any other dimensions to be ignored. Resizing of the window can cause this setting to be automatically negated (see MLN_OVERFLOW).

MLFFMTRECT_LIMITHORZ

The width of any line in the MLE cannot exceed the given horizontal dimension. If word-wrap is on, this limit has no effect. Word-wrap can result in trailing blanks beyond the right limit. These do not cause an overflow notification.

MLFFMTRECT_LIMITVERT

The vertical height of the total text, as displayed, is limited to that which fits totally within the vertical dimension of the format rectangle.

rc (BOOL)

Success indicator.

TRUE

Successful completion

FALSE

An error occurred.

MLM_SETFORMATRECT - Syntax

This message sets the format dimensions and mode.

```
param1
    PPOINTL  pFormatRect /* New format dimensions. */

param2
    ULONG    flFlags      /* Flags governing interpretation of dimensions. */
```

MLM_SETFORMATRECT - Remarks

The multi-line entry field control window procedure responds to this message by setting formatting dimensions and mode.

Any addition of text that causes the text to exceed the rectangle limits causes a notification before proceeding (see MLN_PIXHORZOVERFLOW and MLN_PIXVERTOVERFLOW).

Any activity that would cause the rectangle to be unable to contain the existing text (resize, undo, increasing font size, or word-wrap on or off) is rejected and results in a notification message for information (see MLN_OVERFLOW).

MLM_SETFORMATRECT - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

MLM_SETFORMATRECT - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

MLM_SETIMPORTEXPOR

MLM_SETIMPORTEEXPORT Field - pBuff

pBuff (**PCHAR**)
Transfer buffer.

MLM_SETIMPORTEEXPORT Field - ulLength

ulLength (**ULONG**)
Size of transfer buffer in bytes.

MLM_SETIMPORTEEXPORT Return Value - rc

rc (**BOOL**)
Success indicator.

TRUE	Successful completion
FALSE	An error occurred.

MLM_SETIMPORTEEXPORT - Parameters

pBuff (**PCHAR**)
Transfer buffer.

ulLength (**ULONG**)
Size of transfer buffer in bytes.

rc (**BOOL**)
Success indicator.

TRUE	Successful completion
FALSE	An error occurred.

MLM_SETIMPORTEEXPORT - Syntax

This message sets the current transfer buffer.

```
param1
    PCHAR  pBuff      /* Transfer buffer. */

param2
    ULONG  ulLength   /* Size of transfer buffer in bytes. */
```

MLM_SETIMPORTEXPOR - Remarks

Given a far pointer to a buffer, and the size of the buffer, this message sets it as the current transfer buffer for the MLE. This buffer is used by the [MLM_IMPORT](#) and [MLM_EXPORT](#) messages. The system segment limit must be observed when specifying the buffer size.

MLM_SETIMPORTEXPOR - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

MLM_SETIMPORTEXPOR - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MLM_SETREADONLY

MLM_SETREADONLY Field - usReadOnly

usReadOnly (USHORT)	New read-only value.
TRUE	Read-only mode set.

FALSE
Read-only mode cleared.

MLM_SETREADONLY Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MLM_SETREADONLY Return Value - rc

rc (BOOL)
Previous read-only value.

TRUE
Read-only mode was set.
FALSE
Read-only mode was cleared.

MLM_SETREADONLY - Parameters

usReadOnly (USHORT)
New read-only value.

TRUE
Read-only mode set.
FALSE
Read-only mode cleared.

ulReserved (ULONG)
Reserved value, should be 0.

rc (BOOL)
Previous read-only value.

TRUE
Read-only mode was set.
FALSE
Read-only mode was cleared.

MLM_SETREADONLY - Syntax

This message sets or clears read-only mode.

```
param1
    USHORT  usReadOnly  /* New read-only value. */

param2
    ULONG   ulReserved  /* Reserved value, should be 0. */
```

MLM_SETREADONLY - Remarks

When read-only mode is set, characters typed at the keyboard do not get inserted into the MLE text. The API insertion interface, however, is still functional, as are selection-manipulation activities and copy-to-clipboard operations. This is useful as a means of preventing text modification (such as in a help system), and for providing a minimal blocking printing semaphore.

MLM_SETREADONLY - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

MLM_SETREADONLY - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

MLM_SETSEL

MLM_SETSEL Field - iptAnchor

iptAnchor ([IPT](#))
Insertion point for new anchor point.

MLM_SETSEL Field - iptCursor

iptCursor (IPT)
Insertion point for new cursor point.

MLM_SETSEL Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Selection successfully set
FALSE	An error occurred.

MLM_SETSEL - Parameters

iptAnchor (IPT)
Insertion point for new anchor point.

iptCursor (IPT)
Insertion point for new cursor point.

rc (BOOL)
Success indicator.

TRUE	Selection successfully set
FALSE	An error occurred.

MLM_SETSEL - Syntax

This message sets a selection.

```
param1
    IPT    iptAnchor /* Insertion point for new anchor point. */
param2
    IPT    iptCursor /* Insertion point for new cursor point. */
```

MLM_SETSEL - Remarks

This message sets the anchor and cursor points. The screen display is updated appropriately, ensuring that the cursor point is visible (which may involve scrolling). Note that the text cursor and inversion are not displayed if the MLE window does not have the input focus. A negative value for a point leaves that point alone.

MLM_SETSEL - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

MLM_SETSEL - Examples

This example highlights the second, third, and fourth characters of the text, and places the cursor to the right of the fourth character.

```
WinSendMsg(hwndMle, MLM_SETSEL, (MPARAM) 1L, (MPARAM) 4L);
```

MLM_SETSEL - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Examples](#)
- [Glossary](#)

MLM_SETTABSTOP

MLM_SETTABSTOP Field - pixTab

pixTab ([PIX](#))

PeI interval for tab stops.

MLM_SETTABSTOP Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

MLM_SETTABSTOP Return Value - pixTabset

pixTabset (PIX)
Success indicator.

< 0	An error occurred.
Other	The value to which the width was set.

MLM_SETTABSTOP - Parameters

pixTab (PIX)
Pel interval for tab stops.

ulReserved (ULONG)
Reserved value, should be 0.

pixTabset (PIX)
Success indicator.

< 0	An error occurred.
Other	The value to which the width was set.

MLM_SETTABSTOP - Syntax

This message sets the pel interval at which tab stops are placed.

```
param1
    PIX    pixTab    /* Pel interval for tab stops. */

param2
    ULONG  ulReserved /* Reserved value, should be 0. */
```

MLM_SETTABSTOP - Remarks

This message fails if the reserved value is not 0.

This message can cause an overflow, see MLN_OVERFLOW.

MLM_SETTABSTOP - Default Processing

The default window procedure takes no action on this message, other than to set *pixTabset* to 0.

MLM_SETTABSTOP - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MLM_SETTEXTCOLOR

MLM_SETTEXTCOLOR Field - IColor

IColor ([LONG](#))
Color.

MLM_SETTEXTCOLOR Field - ITextColor

ITextColor ([LONG](#))
Text color flag.

This flag can be set to one of the following:

MLE_INDEX	Indexed (solid) colors.
	For version 3, or lower, of the OS/2 operating system the default is MLE_INDEX=0; MLE does not support dithered colors.
MLE_RGB	Dithered colors.

For versions, higher than version 3, of the OS/2 operating system colors default to the RGB mode.

MLM_SETTEXTCOLOR Return Value - IOldColor

IOldColor ([LONG](#))
Color previously used.

MLM_SETTEXTCOLOR - Parameters

IColor ([LONG](#))
Color.

ITextColor ([LONG](#))
Text color flag.

This flag can be set to one of the following:

MLE_INDEX Indexed (solid) colors.

For version 3, or lower, of the OS/2 operating system the default is MLE_INDEX=0; MLE does not support dithered colors.

MLE_RGB Dithered colors.

For versions, higher than version 3, of the OS/2 operating system colors default to the RGB mode.

IOldColor ([LONG](#))
Color previously used.

MLM_SETTEXTCOLOR - Syntax

This message sets the text color.

```
param1
    LONG  IColor      /* Color. */

param2
    LONG  ITextColor  /* Text color flag. */
```

MLM_SETTEXTCOLOR - Remarks

This message sets the color in which the MLE text is to be drawn, and updates the display as necessary.
The color values are the same as those used by GpiSetColor.

MLM_SETTEXTCOLOR - Default Processing

The default window procedure takes no action on this message, other than to set *OldColor* to 0.

MLM_SETTEXTCOLOR - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MLM_SETTEXTLIMIT

MLM_SETTEXTLIMIT Field - ISize

ISize ([LONG](#))
Maximum number of characters in MLFIE_NOTRANS format.

MLM_SETTEXTLIMIT Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_SETTEXTLIMIT Return Value - ulFit

ulFit (**ULONG**)
Success indicator.

0
Successful completion. Current text fits within the new limit.

Other
The number of bytes by which the current text exceeds the proposed limit. The limit is not changed.

MLM_SETTEXTLIMIT - Parameters

lSize (**LONG**)
Maximum number of characters in MLFIE_NOTRANS format.

ulReserved (**ULONG**)
Reserved value, should be 0.

ulFit (**ULONG**)
Success indicator.

0
Successful completion. Current text fits within the new limit.

Other
The number of bytes by which the current text exceeds the proposed limit. The limit is not changed.

MLM_SETTEXTLIMIT - Syntax

This message sets the maximum number of bytes that a multi-line entry field control can contain.

```
param1  
    LONG    lSize        /* Maximum number of characters in MLFIE_NOTRANS format. */  
  
param2  
    ULONG    ulReserved  /* Reserved value, should be 0. */
```

MLM_SETTEXTLIMIT - Remarks

The multi-line entry field control window procedure responds to this message by limiting the text size to *lSize* bytes. Text size is calculated using the MLFIE_NOTRANS format. Note that this is bytes and *not* characters; DBCS programmers should calculate accordingly.

This message returns 0 if the text limit exceeds or is equal to the existing text. Otherwise it returns the number of bytes by which the text would have overflowed, and does not change the limit.

The default, which is unbounded, can be specified by entering a non-positive limit.

MLM_SETTEXTLIMIT - Default Processing

The default window procedure takes no action on this message, other than to set *ulFit* to 0.

MLM_SETTEXTLIMIT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MLM_SETWRAP

MLM_SETWRAP Field - usWrap

usWrap ([USHORT](#))
New value for wrap flag.

MLM_SETWRAP Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_SETWRAP Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	An error occurred.

MLM_SETWRAP - Parameters

usWrap (USHORT)

New value for wrap flag.

ulReserved (ULONG)

Reserved value, should be 0.

rc (BOOL)

Success indicator.

TRUE

Successful completion

FALSE

An error occurred.

MLM_SETWRAP - Syntax

This message sets the wrap flag.

```
param1
    USHORT    usWrap      /* New value for wrap flag. */

param2
    ULONG     ulReserved  /* Reserved value, should be 0. */
```

MLM_SETWRAP - Remarks

The multi-line entry field control window procedure responds to this message by setting the word wrap mode and updating the screen as appropriate.

When word-wrap is turned on, the text is wrapped to fit the formatting rectangle width. When word-wrap is turned off, the text is allowed to trail off to the right until it reaches an end-of-line marker.

Word-wrapping is defined as follows. Words are sequences of non-white-space characters (white-space characters are space, line break, and tab). When word-wrapping is enabled, the whole word must appear on one line within the formatting rectangle, unless the word by itself is too long to fit. In this case the word is split following the last character that fits, and the remainder starts a new line.

This definition then applies recursively to the remainder of the word. The word continues to be visible. For editing purposes (for example, for word-selection) the word is viewed as a single word drawn over multiple lines.

Blank characters are always accumulated onto the current line, even if they exceed the horizontal formatting dimension, that is, blanks are allowed to trail off the right-hand edge. Line-break characters are also allowed to exceed the horizontal dimension, and any subsequent text must begin on a new line. The line-break following a line-break character is sometimes referred to as a hard line-break. Other line breaks, due to word-wrapping, and not to explicit formatting characters, are referred to as soft line-breaks.

Tab characters must always be visible. If a tab character occurs after the last tab stop within the horizontal formatting dimension, a soft line-break occurs after the tab.

This message can cause an overflow, see MLN_OVERFLOW.

MLM_SETWRAP - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

MLM_SETWRAP - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

MLM_UNDO

MLM_UNDO Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_UNDO Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

MLM_UNDO Return Value - rc

rc ([USHORT](#))
Success indicator.

TRUE
An undo operation was performed.

FALSE

No undo operation was performed.

MLM_UNDO - Parameters

ulReserved (ULONG)

Reserved value, should be 0.

ulReserved (ULONG)

Reserved value, should be 0.

rc (USHORT)

Success indicator.

TRUE

An undo operation was performed.

FALSE

No undo operation was performed.

MLM_UNDO - Syntax

This message performs any available undo operation.

```
param1
    ULONG    ulReserved /* Reserved value, should be 0. */
param2
    ULONG    ulReserved /* Reserved value, should be 0. */
```

MLM_UNDO - Remarks

The last operation is undone (note that an undo can be undone).

This can cause an overflow, see MLN_OVERFLOW.

MLM_UNDO - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

MLM_UNDO - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

WM_BUTTON1DBLCLK (in Multiline Entry Fields)

WM_BUTTON1DBLCLK (in Multiline Entry Fields) - Syntax

For the cause of this message, see [WM_BUTTON1DBLCLK](#).

For a description of the parameters, see [WM_BUTTON1DBLCLK](#).

WM_BUTTON1DBLCLK (in Multiline Entry Fields) - Remarks

This message indicates that mouse button 1 has clicked twice within the system double-click time.

Double-Click

If the click point is in the middle of a non-white-space character, the token (word) surrounding the clicked-on character, and any trailing spaces, are selected. If the click point is in a space character, the previous word (along with the trailing spaces including the clicked-on space) is selected. If there is no preceding word (either because the spaces are at the beginning of the text or immediately follow a line-break character) the run of spaces is selected. If the click point is on a tab or line-break character, that character is selected.

Shift-Double-Click

Double-clicking while the Shift key is pressed leaves the anchor point alone, and moves the cursor point to the beginning or end of the clicked-on token. If the click point is before the anchor point in the text, the cursor point is moved to the beginning of the surrounding word, otherwise, the cursor point is moved to the end of the surrounding word. When shift-double-clicking, the selection is extended to include the token that was double-clicked on.

Margin Mouse Event

All mouse events in a margin cause the MLE to send a MLN_MARGIN notification to the owner window of the MLE. This message has, as its parameters, the original mouse message. The owner can process the notification or not. If the owner does not process the message, the event is treated as if it occurred on the closest point in the text.

WM_BUTTON1DBLCLK (in Multiline Entry Fields) - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_BUTTON1DBLCLK (in Multiline Entry Fields) - Related Messages

Related Messages

- [WM_BUTTON1DBLCLK](#)

WM_BUTTON1DBLCLK (in Multiline Entry Fields) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_BUTTON1DOWN (in Multiline Entry Fields)

WM_BUTTON1DOWN (in Multiline Entry Fields) - Syntax

For the cause of this message, see [WM_BUTTON1DOWN](#).

For a description of the parameters, see [WM_BUTTON1DOWN](#).

WM_BUTTON1DOWN (in Multiline Entry Fields) - Remarks

This message delimits mouse button click events. Between a button-down and a button-up event, the mouse is considered to be dragging. A mouse click is considered to happen on button-down, and dragging is terminated by a button-up.

Click

Clicking in the text sets the cursor and anchor points to the nearest insertion point. If the MLE is in overtype mode, the anchor is extended one character further in the text, subject to the end-of-text and new-line boundary conditions, defined under [WM_CHAR](#) (in [Multiline Entry Fields](#)).

Shift-Click

Clicking while the shift key is held down sets the cursor point to the nearest insertion point, while leaving the anchor point alone.

Margin Mouse Event

All mouse events in a margin cause the MLE to send a MLN_MARGIN notification to the owner window of the MLE. This message has, as its parameters, the original mouse message. The owner can process the notification or not. If the owner does not process the message, the event is treated as if it occurred on the closest point in the text.

WM_BUTTON1DOWN (in Multiline Entry Fields) - Default Processing

The default window procedure takes no action on this message, other than to set `rc` to FALSE.

WM_BUTTON1DOWN (in Multiline Entry Fields) - Related Messages

Related Messages

- [WM_BUTTON1DOWN](#)
-

WM_BUTTON1DOWN (in Multiline Entry Fields) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_BUTTON1UP (in Multiline Entry Fields)

WM_BUTTON1UP (in Multiline Entry Fields) - Syntax

For the cause of this message, see [WM_BUTTON1UP](#).

For a description of the parameters, see [WM_BUTTON1UP](#).

WM_BUTTON1UP (in Multiline Entry Fields) - Remarks

This message delimits mouse button click events. Between a button-down and a button-up event the mouse is considered to be dragging. A mouse click is considered to happen on button-down, and dragging is terminated by a button-up.

Margin Mouse Event

All mouse events in a margin cause the MLE to send a MLN_MARGIN notification to the owner window of the MLE. This message has, as its parameters, the original mouse message. The owner can process the notification or not. If the owner does not process the message, the event is treated as if it occurred on the closest point in the text.

WM_BUTTON1UP (in Multiline Entry Fields) - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_BUTTON1UP (in Multiline Entry Fields) - Related Messages

Related Messages

- [WM_BUTTON1UP](#)

WM_BUTTON1UP (in Multiline Entry Fields) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_CHAR (in Multiline Entry Fields)

WM_CHAR (in Multiline Entry Fields) - Syntax

For the cause of this message, see [WM_CHAR](#).

For a description of the parameters, see [WM_CHAR](#).

WM_CHAR (in Multiline Entry Fields) - Remarks

The behavior of the MLE, when typing, depends on whether it is in insert or overwrite mode, and whether the selection is empty or not. The selection is defined to be empty when the cursor point is equal to the anchor point.

When a character is typed, it replaces the current selection. If the selection is empty, the character is viewed as replacing nothing, so the character is effectively inserted into the text. If one or more characters are selected, those characters are deleted from the text and replaced by the typed character.

If the MLE is in insert mode, the cursor and anchor points are moved to immediately follow the newly typed character.

If the MLE is in overwrite mode, the cursor is moved to immediately follow the newly typed character. If there is no character after the cursor (the new character is at the end of the text) or if the character after the cursor is a line-break character, the anchor is set to be equal to the cursor point. In any other case, the anchor is extended one character past the cursor point, defining the next character as the current selection.

If the typing causes the cursor to go off the screen in any direction, the display is automatically scrolled. If word-wrap is on, text continues on a new line, otherwise, the screen is scrolled horizontally.

Scrolling of the text in the window is independent of cursor movement. The cursor and selection remain unaltered at the same location within the text during all scrolling but the converse is not true. Any movement of the cursor causes auto-scrolling, if necessary, to ensure that the text location of the cursor is visible within the window.

Tabs

Tabs are represented as a single character in the text model, and are displayed as enough white-space to reach the next tab stop. Tab stops are set at pel intervals, starting with zero and occurring every n pels, where n is a value set by the [MLM_SETTABSTOP](#) message, and defaulting to eight times the average character width of the system font. When a tab is drawn, it uses the number of pels defined by the following formula:

$$\text{pelWidth} = \text{pelTab} - (\text{pelDraw} \bmod \text{pelTab})$$

where pelTab is the tab interval, in pels, and pelDraw is the pel at which drawing is to begin.

Return

Return (ASCII newline) causes a hard line-break, and the following text begins on a new line. A line-break character is inserted in the text, which is drawn as a few pels of white-space (for selection purposes).

Keystroke commands

For all the following keys, unless otherwise noted, the display is scrolled, if necessary, to keep the cursor point visible. Where noted, the cursor setting behaves differently in insert mode than in overwrite mode. This is subject to the boundary conditions noted above.

Del

Causes the contents of the selection region to be deleted. If the selection region contains no text, it causes the character to the right of the cursor to be deleted.

Shift+Del

Causes the contents of the selection region to be cut to the clipboard.

Insert

Toggles between insert and overwrite mode. The MLE ignores the Insert key when it occurs without a modifier.

Shift+Ins

Causes the contents of the clipboard to replace the selection region.

Ctrl+Ins

Causes the selection region to be copied to the clipboard. The selection region is not otherwise affected.

Backspace

Functions similar to Del. If the selection is not empty, Backspace deletes the selection. If the selection is empty, Backspace deletes the character to the left of the cursor point. If the MLE is in overwrite mode, the anchor point is set, and the cursor point is moved to be one character previous in the text. If no such character exists (because the anchor is set to the beginning of the text) the cursor is set to the anchor point. If the MLE is in insert mode, the cursor and anchor points are set, as defined at the start of this chapter.

Down Arrow

Sets the cursor point to the closest insertion point on the following line, then sets the anchor point to the cursor point (insertion mode) or one character following (overwrite mode).

Shift+Down Arrow

Causes the cursor point to be moved to the closest insertion point on the following line. The anchor point does not move.

Up Arrow

Sets the cursor point to the closest insertion point on the preceding line, then sets the anchor point to the cursor point (insert mode) or one character following (overtyping mode).

Shift+Up

Sets the cursor point to the closest insertion point on the preceding line. The anchor point is not moved.

Right Arrow

Sets the cursor point to the insertion point one character following the cursor point. The anchor point is set to the cursor point (insert mode) or one character following (overtyping mode).

Shift+Right

Causes the cursor point to be set to the insertion point immediately following the previous cursor point. The anchor point is not moved.

Left or Shift+Left

Work analogously.

Ctrl+Right

Moves the cursor point to the insertion point immediately preceding the next word in the text including trailing spaces, and sets the anchor point to be equal to (insert mode) or one character following (overtyping mode) the cursor point. The EOL (hard line-break) and tab characters are treated as words.

Ctrl+Shift+Right

Moves only the cursor point in the same way as Ctrl+Right, but leaves the anchor point unmoved.

Ctrl+Left

Moves the cursor point to the preceding insertion point at the beginning of a word, and sets the anchor point to be equal to (insert mode) or one character following (overtyping mode) the cursor point. The EOL (hard line-break) and tab characters are treated as words.

Ctrl+Shift+Left

Moves only the cursor point in the same way as Ctrl+Left but leaves the anchor point unmoved.

PageDown or PageUp

Cause the display to be scrolled one screen at a time in either direction. This behavior is the same as would be encountered during a page-down or page-up caused by the scroll-bar.

Ctrl+PageDown or Ctrl+PageUp

Cause the display to be scrolled one screen at a time to the right or left respectively. This behavior is the same as would be encountered during a page-right or page-left caused by the scroll-bar.

Home

Sets the cursor point to the insertion point at the beginning of the line containing the cursor point, and sets the anchor point equal to (insert mode) or one character following (overtyping mode).

Shift+Home

Moves the cursor point to the insertion point at the beginning of the line. The anchor point is not moved.

End

Sets the anchor point to the insertion point at the end of the line containing the cursor point. If the last character on the line is a line-break character, the anchor is positioned just before it. The cursor is set equal to (insert mode) or one character previous to (overtyping mode) the anchor.

Shift+End

Moves the cursor point to the insertion point at the end of the line, as above. The anchor point is not moved.

Ctrl+Home

Moves the cursor point to the insertion point at the beginning of the document. The anchor point is set equal to (insert mode) or one character following it (overtyping mode).

Ctrl+End

Moves the anchor point to the insertion point at the end of the document. The cursor point is set to be equal to the anchor point (insert mode) or one character preceding it (overtyping mode).

Ctrl+Shift+Home

Moves the cursor point in the same way as Ctrl+Home, but leaves the anchor point unmoved.

Ctrl+Shift+End

Moves the cursor point in the same way as Ctrl+End, but leaves the anchor point unmoved.

WM_CHAR (in Multiline Entry Fields) - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_CHAR (in Multiline Entry Fields) - Related Messages

Related Messages

- [WM_CHAR](#)
-

WM_CHAR (in Multiline Entry Fields) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_ENABLE (in Multiline Entry Fields)

WM_ENABLE (in Multiline Entry Fields) - Syntax

For the cause of this message, see [WM_ENABLE](#).

For a description of the parameters, see [WM_ENABLE](#).

WM_ENABLE (in Multiline Entry Fields) - Remarks

The multi-line entry field control window procedure responds to this message by setting the enable state and by setting *ulReserved* to 0.

Disabling the window is similar, but not identical, to [MLM_DISABLELREFRESH](#). Enabling the window is similar, but not identical, to [MLM_ENABLELREFRESH](#). (Note that this also applies to window styles.) The difference is that a disabled window receives no mouse or keyboard input whereas with [MLM_DISABLELREFRESH](#) it receives the input but discards it.

WM_ENABLE (in Multiline Entry Fields) - Default Processing

The default window procedure takes no action on this message, other than to set *u/Reserved* to 0.

WM_ENABLE (in Multiline Entry Fields) - Related Messages

Related Messages

- [WM_ENABLE](#)
-

WM_ENABLE (in Multiline Entry Fields) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_MOUSEMOVE (in Multiline Entry Fields)

WM_MOUSEMOVE (in Multiline Entry Fields) - Syntax

For the cause of this message, see [WM_MOUSEMOVE](#).

For a description of the parameters, see [WM_MOUSEMOVE](#).

WM_MOUSEMOVE (in Multiline Entry Fields) - Remarks

The mouse pointer moves and is of interest to the MLE. If refresh is disabled, the pointer is set to the wait icon (a clock face). If refresh is enabled, the pointer is set to an I-beam. This message can occur during dragging or when simply tracking the mouse.

Dragging

Dragging sets the selection anchor to be the point where dragging begins, and moves the cursor point along with it as the mouse is moved. Moving the pointer into the margins while dragging produces a scroll in the appropriate direction and continues selecting.

Margin Mouse Event

All mouse events in a margin cause the MLE to send a `MLN_MARGIN` notification to the owner window MLE. This message has, as its parameters, the original mouse message. The owner can process the notification or not. If the owner does not process the message, the event is treated as if it occurred on the closest point in the text.

WM_MOUSEMOVE (in Multiline Entry Fields) - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to 0 (FALSE).

WM_MOUSEMOVE (in Multiline Entry Fields) - Related Messages

Related Messages

- [WM_MOUSEMOVE](#)
-

WM_MOUSEMOVE (in Multiline Entry Fields) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_QUERYWINDOWPARAMS (in Multiline Entry Fields)

WM_QUERYWINDOWPARAMS (in Multiline Entry Fields) - Syntax

This message occurs when an application queries the entry field control window parameters.

For a description of the parameters, see [WM_QUERYWINDOWPARAMS](#).

WM_QUERYWINDOWPARAMS (in Multiline Entry Fields) - Remarks

The multi-line entry field control window procedure responds to this message by returning the window parameters indicated by the *fsStatus* parameter of the [WNDPARAMS](#) data structure, identified by the *pwndparams* parameter.

In response to the WPM_CCHTEXT flag, the text length is reported in the CF_TEXT format. If it exceeds 64KB-1, then this value is reported. In response to the WPM_TEXT flag, text up to the amount returned for the WPM_CCHTEXT value is placed at the indicated location in CF_TEXT format.

WM_QUERYWINDOWPARAMS (in Multiline Entry Fields) - Default Processing

The default window procedure sets the *cchText*, *cbPresParams*, and *cbCtlData* parameters of the [WNDPARAMS](#) data structure, identified by *pwndparams*, to 0 and sets *rc* to FALSE.

WM_QUERYWINDOWPARAMS (in Multiline Entry Fields) - Related Messages

Related Messages

- [WM_QUERYWINDOWPARAMS](#)

WM_QUERYWINDOWPARAMS (in Multiline Entry Fields) - Topics

Select an item:
[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_SETWINDOWPARAMS (in Multiline Entry Fields)

WM_SETWINDOWPARAMS (in Multiline Entry Fields) - Syntax

This message occurs when an application sets or changes the entry field control window parameters.

For a description of the parameters, see [WM_SETWINDOWPARAMS](#).

WM_SETWINDOWPARAMS (in Multiline Entry Fields) - Remarks

The multi-line entry field control window procedure responds to this message by setting the window parameters indicated by the *fsStatus* parameter of the [WNDPARAMS](#) data structure, identified by the *pwndparams* parameter.

If the MLE text is to be set by this message, it is assumed to be in CF_TEXT format (see [MLM_FORMAT](#)) and all existing text is deleted before the new text is inserted. Note that a Control Data structure can be associated with the window parameters, in which case any field in that structure can cause a change to the MLE.

WM_SETWINDOWPARAMS (in Multiline Entry Fields) - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_SETWINDOWPARAMS (in Multiline Entry Fields) - Related Messages

Related Messages

- [WM_SETWINDOWPARAMS](#)
-

WM_SETWINDOWPARAMS (in Multiline Entry Fields) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

Notebook Control Window Processing

This system-provided window procedure processes the actions on a notebook control (WC_NOTEBOOK).

Purpose

A notebook control (WC_NOTEBOOK window class) is a visual component whose specific purpose is to organize information on individual pages so that a user can find and display that information quickly and easily. It simulates a real-world notebook while improving it by overcoming its natural limitations. A user can select and display pages by using either a pointing device, such as a mouse, or the keyboard.

The notebook is designed to be customizable to meet varying application requirements, while providing an easy-to-use user interface component that can be used to develop products that conform to the Common User Access* (CUA*) user interface guidelines. The application can specify different colors, sizes, and orientations for its notebooks, and whether the notebook should be the old or new style, but the underlying function of the control remains the same.

Old Notebook Control Styles

Old notebook control window styles can be set with a notebook is created. The following styles can be set when creating a notebook control window. If no styles are specified, defaults, which are identified in the following descriptions, are used.

- Specify one of the following to determine whether the control is a a solid bound or spiral bound notebook:

BKS_SOLIDBIND	Paints a solid binding on the notebook. This is the default.
BKS_SPIRALBIND	Paints a spiral binding on the notebook.
- Specify one of the following to determine where the back pages are positioned:

BKS_BACKPAGESBR	Paints back pages on the notebook's bottom and right sides. This is the default.
BKS_BACKPAGESBL	Paints back pages on the notebook's bottom and left sides.
BKS_BACKPAGESTR	Paints back pages on the notebook's top and right sides.
BKS_BACKPAGESTL	Paints back pages on the notebook's top and left sides.
- Specify one of the following to determine the side of the notebook on which the major tabs are positioned. Valid combinations with back pages styles are noted in each definition.

BKS_MAJORTABRIGHT	Places major tabs on the notebook's right edge. Only valid in combination with BKS_BACKPAGESBR or BKS_BACKPAGESTR. This is the default when either of these back pages styles is used.
BKS_MAJORTABLEFT	Places major tabs on the notebook's left edge. Only valid in combination with BKS_BACKPAGESBL or BKS_BACKPAGESTL. This is the default when BKS_BACKPAGESTL is used.
BKS_MAJORTABTOP	Places major tabs on the notebook's top edge. Only valid in combination with BKS_BACKPAGESTR or BKS_BACKPAGESTL.
BKS_MAJORTABBOTTOM	Places major tabs on the notebook's bottom edge. Only valid in combination with BKS_BACKPAGESBR or BKS_BACKPAGESBL. This is the default when BKS_BACKPAGESBL is used.
- Specify one of the following to set the shape of the notebook tabs:

BKS_SQUARETABS	Draws tabs with square edges. This is the default.
BKS_ROUNDEDTABS	Draws tabs with rounded edges.
BKS_POLYGONTABS	Draws tabs with polygon edges.
- Specify one of the following to position the status line text:

BKS_STATUSTEXTLEFT	Left-justifies status line text. This is the default.
BKS_STATUSTEXTRIGHT	Right-justifies status line text.
BKS_STATUSTEXTCENTER	Centers status line text.
- Specify one of the following to position the tab text:

BKS_TABTEXTCENTER	Centers tab text. This is the default.
BKS_TABTEXTLEFT	Left-justifies tab text.
BKS_TABTEXTRIGHT	Right-justifies tab text.
<ul style="list-style-type: none"> The following styles are not valid with an old notebook: 	
BKS_TABBEDDIALOG	Indicates a new style notebook.
BKS_BUTTONAREA	Creates a common button area in a new style notebook. This style uses the same area as BKS_POLYGONTABS, and will cause unexpected results if used in an old notebook.

New Notebook Control Styles

New notebook control window styles can be set when a notebook is created. A new notebook is indicated by the BKS_TABBEDDIALOG style being specified. The following styles can be set when creating a notebook control window. If no styles are specified, defaults, which are identified in the following descriptions, are used.

<ul style="list-style-type: none"> Specify the following to determine the position of the major tabs on the new notebook and the ordering of the tabs in the notebook. 		
BKS_MAJORTABTOP	Places major tabs on the notebook's top edge. This is the default for a new notebook.	
	BKS_BACKPAGESTR	Tabs are ordered left to right (default)
	BKS_BACKPAGESTL	Tabs are ordered right to left.
BKS_MAJORTABBOTTOM	Places major tabs on the notebook's bottom edge.	
	BKS_BACKPAGESBR	Tabs are ordered left to right (default)
	BKS_BACKPAGESBL	Tabs

- Specify a common button area:

BKS_BUTTONAREA

Creates a common button area in a new style notebook.
- The following styles are ignored for a new notebook:

BKS_SOLIDBIND

Paints a solid binding on the notebook. This is the default.

BKS_SPIRALBIND

Paints a spiral binding on the notebook.

BKS_MAJORTABRIGHT

Places major tabs on the notebook's right edge. This is the default when either of these back pages styles is used.

BKS_MAJORTABLEFT

Places major tabs on the notebook's left edge.

BKS_SQUARETABS

Draws tabs with square edges.

BKS_ROUNDEDTABS

Draws tabs with rounded edges.

BKS_POLYGONTABS

Draws tabs with polygon edges.

BKS_TABTEXTCENTER

Centers tab text.

BKS_TABTEXTLEFT

Left-justifies tab text.

BKS_TABTEXTRIGHT

Right-justifies tab text.

BKS_STATUSTEXTLEFT

Left-justifies status line text.

BKS_STATUSTEXTRIGHT

Right-justifies status line text.

BKS_STATUSTEXTCENTER

Centers status line text.

Notebook Control Data

See the following for descriptions of the notebook control data structures:

- [BOOKTEXT](#)
- [DELETENOTIFY](#)
- [PAGESELECTNOTIFY](#)
- [NOTEBOOKBUTTON.](#)

Notebook Control Notification Messages

These messages are initiated by the notebook control window to notify its owner of significant events.

WM_CONTROL (in Notebook Controls)

WM_CONTROL (in Notebook Controls) Field - id

id ([USHORT](#))

Control-window identity.

WM_CONTROL (in Notebook Controls) Field - notifycode

notifycode ([USHORT](#))

Notify code.

The notebook control uses these notification codes:

BKN_HELP

Indicates the notebook control has received a [WM_HELP](#) message.

BKN_NEWPAGESIZE

Indicates the dimensions of the application page window have changed.

BKN_PAGEDELETED

Indicates a page has been deleted from the notebook.

BKN_PAGESELECTED

Indicates a new page has been brought to the top of the notebook. This notification is sent after the page is turned.

BKN_PAGESELECTEDPENDING

Indicates a new page is about to be brought to the top of the notebook. This notification is sent before the page is actually turned.

If the application does not want the page to be turned, it sets the *ulPageIdNew* field of the [PAGESELECTNOTIFY](#) structure to NULL before returning.

WM_CONTROL (in Notebook Controls) Field - notifyinfo

notifyinfo ([ULONG](#))

Notify code information.

The value of this parameter depends on the value of the *notifycode* parameter. When the value of the *notifycode* parameter is BKN_HELP, this parameter is the ID of the notebook page (*ulPageId*) whose tab contains the selection cursor.

When the value of the *notifycode* parameter is BKN_PAGESELECTED or BKN_PAGESELECTEDPENDING, this parameter is a pointer to the [PAGESELECTNOTIFY](#) structure.

When the value of the *notifycode* parameter is BKN_PAGEDELETED, this parameter is a pointer to the [DELETENOTIFY](#) structure.

Otherwise, this parameter is the notebook control window handle.

WM_CONTROL (in Notebook Controls) Return Value - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_CONTROL (in Notebook Controls) - Parameters

id ([USHORT](#))

Control-window identity.

notifycode ([USHORT](#))

Notify code.

The notebook control uses these notification codes:

BKN_HELP

Indicates the notebook control has received a [WM_HELP](#) message.

BKN_NEWPAGESIZE

Indicates the dimensions of the application page window have changed.

BKN_PAGEDELETED

Indicates a page has been deleted from the notebook.

BKN_PAGESELECTED

Indicates a new page has been brought to the top of the notebook. This notification is sent after the page is turned.

BKN_PAGESELECTEDPENDING

Indicates a new page is about to be brought to the top of the notebook. This notification is sent before the page is actually turned.

If the application does not want the page to be turned, it sets the *ulPageIdNew* field of the [PAGESELECTNOTIFY](#) structure to NULL before returning.

notifyinfo ([ULONG](#))

Notify code information.

The value of this parameter depends on the value of the *notifycode* parameter. When the value of the *notifycode* parameter is **BKN_HELP**, this parameter is the ID of the notebook page (*ulPageId*) whose tab contains the selection cursor.

When the value of the *notifycode* parameter is **BKN_PAGESELECTED** or **BKN_PAGESELECTEDPENDING**, this parameter is a pointer to the [PAGESELECTNOTIFY](#) structure.

When the value of the *notifycode* parameter is **BKN_PAGEDELETED**, this parameter is a pointer to the [DELETENOTIFY](#) structure.

Otherwise, this parameter is the notebook control window handle.

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_CONTROL (in Notebook Controls) - Syntax

For the cause of this message, see [WM_CONTROL](#).

```
param1
    USHORT id          /* Control-window identity. */
    USHORT notifycode  /* Notify code. */

param2
    ULONG notifyinfo  /* Notify code information. */
```

WM_CONTROL (in Notebook Controls) - Remarks

The notebook control window procedure generates this message and sends it to its owner, informing the owner of this event.

WM_CONTROL (in Notebook Controls) - Default Processing

For a description of the default processing, see [WM_CONTROL](#).

WM_CONTROL (in Notebook Controls) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_CONTROLPOINTER (in Notebook Controls)

WM_CONTROLPOINTER (in Notebook Controls) - Syntax

For the cause of this message, see [WM_CONTROLPOINTER](#).

For a description of the parameters, see [WM_CONTROLPOINTER](#).

WM_CONTROLPOINTER (in Notebook Controls) - Remarks

For the appropriate remarks, see [WM_CONTROLPOINTER](#).

WM_CONTROLPOINTER (in Notebook Controls) - Default Processing

For the default processing, see [WM_CONTROLPOINTER](#).

WM_CONTROLPOINTER (in Notebook Controls) - Topics

- Select an item:
- [Syntax](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_DRAWITEM (in Notebook Controls)

WM_DRAWITEM (in Notebook Controls) Field - id

id ([USHORT](#))
Window identifier.
The window identifier of the notebook control sending this notification message.

WM_DRAWITEM (in Notebook Controls) Field - powneritem

powneritem ([POWNERITEM](#))
Pointer to an [OWNERITEM](#) data structure.
The following list defines the [OWNERITEM](#) data structure fields that apply to the notebook control.

<i>hwnd</i> (HWND)	Notebook window handle.
<i>hps</i> (HPS)	Presentation-space handle.
<i>fsState</i> (ULONG)	Notebook window style flags. See Notebook Styles for an Old Notebook and Notebook Styles for a New Notebook

for descriptions of these style flags.

fsAttribute ([ULONG](#))
 Page attribute flags for the tab page. See [BKM_INSERTPAGE](#) for descriptions of these attribute flags.

fsStateOld ([ULONG](#))
 Reserved.

fsAttributeOld ([ULONG](#))
 Reserved.

rclItem ([RECTL](#))
 Tab rectangle to be drawn in window coordinates.

idlItem ([LONG](#))
 Reserved.

hlItem ([ULONG](#))
 Current page ID (*ulPageId*) for which the content of a tab is to be drawn.

WM_DRAWITEM (in Notebook Controls) Return Value - rc

rc ([BOOL](#))
 Content-drawn indicator.

TRUE
 The owner draws the tab's content.

FALSE
 If the owner does not draw the tab's content, the owner returns this value and the notebook control draws the tab's content.

WM_DRAWITEM (in Notebook Controls) - Parameters

id ([USHORT](#))
 Window identifier.

The window identifier of the notebook control sending this notification message.

powneritem ([POWNERITEM](#))
 Pointer to an [OWNERITEM](#) data structure.

The following list defines the [OWNERITEM](#) data structure fields that apply to the notebook control.

hwnd ([HWND](#))
 Notebook window handle.

hps ([HPS](#))
 Presentation-space handle.

fsState ([ULONG](#))
 Notebook window style flags. See [Notebook Styles for an Old Notebook](#) and [Notebook Styles for a New Notebook](#) for descriptions of these style flags.

fsAttribute ([ULONG](#))
 Page attribute flags for the tab page. See [BKM_INSERTPAGE](#) for descriptions of these attribute flags.

fsStateOld ([ULONG](#))
 Reserved.

fsAttributeOld ([ULONG](#))
 Reserved.

rclItem ([RECTL](#))
 Tab rectangle to be drawn in window coordinates.

idlItem ([LONG](#))
 Reserved.

hlItem ([ULONG](#))
 Current page ID (*ulPageId*) for which the content of a tab is to be drawn.

rc ([BOOL](#))

Content-drawn indicator.

TRUE

The owner draws the tab's content.

FALSE

If the owner does not draw the tab's content, the owner returns this value and the notebook control draws the tab's content.

WM_DRAWITEM (in Notebook Controls) - Syntax

This notification message is sent to the owner of a notebook control each time a tab's content is to be drawn by the owner of the notebook. The tab's content is drawn by the owner unless the owner sets the tab text or bit map by sending a [BKM_SETTABTEXT](#) or [BKM_SETTABBITMAP](#) message, respectively, to the notebook control.

```
param1
    USHORT        id           /* Window identifier. */

param2
    POWNERITEM    powneritem   /* Pointer to an OWNERITEM data structure. */
```

WM_DRAWITEM (in Notebook Controls) - Remarks

If an application uses notebook controls that contain tab pages, the default condition is for the application to draw the contents of the tab each time a tab page is displayed. This situation applies particularly if the content of the tab is not one of the supported formats.

The notebook control window procedure generates this message and sends it to its owner, informing the owner that the content of a tab is to be drawn. The owner is given the opportunity to draw the content of the tab and to indicate that the content of the tab has been drawn or that the notebook control is to draw it. To indicate that the notebook control is to draw the content of the tab, the owner sends either a [BKM_SETTABTEXT](#) or a [BKM_SETTABBITMAP](#) message to the notebook control.

WM_DRAWITEM (in Notebook Controls) - Default Processing

For a description of the default processing, see [WM_DRAWITEM](#).

WM_DRAWITEM (in Notebook Controls) - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

Notebook Control Window Messages

This section describes the notebook control window procedure actions on receiving the following messages.

BKM_CALCPAGERECT

BKM_CALCPAGERECT Field - pRectl

pRectl ([PRECTL](#))

Pointer to the [RECTL](#) structure that contains the coordinates of the rectangle.

If the *bPage* parameter is TRUE, this structure contains the coordinates of a notebook window on input, and on return it contains the coordinates of an application page window.

If the *bPage* parameter is FALSE, this structure contains the coordinates of an application page window on input, and on return it contains the coordinates of a notebook window.

BKM_CALCPAGERECT Field - bPage

bPage ([BOOL](#))

Window specifier.

Specifies whether the window coordinates to calculate are for a notebook window or an application page window.

TRUE

An application page window is calculated.

FALSE

A notebook window is calculated.

BKM_CALCPAGERECT Return Value - rc

rc ([BOOL](#))

Success indicator.

TRUE

Coordinates were successfully calculated.

FALSE

Unable to calculate coordinates. This is returned if an invalid [RECTL](#) structure is specified in the *pRectl* parameter.

BKM_CALCPAGERECT - Parameters

pRectl (RECTL)

Pointer to the RECTL structure that contains the coordinates of the rectangle.

If the *bPage* parameter is TRUE, this structure contains the coordinates of a notebook window on input, and on return it contains the coordinates of an application page window.

If the *bPage* parameter is FALSE, this structure contains the coordinates of an application page window on input, and on return it contains the coordinates of a notebook window.

bPage (BOOL)

Window specifier.

Specifies whether the window coordinates to calculate are for a notebook window or an application page window.

TRUE

An application page window is calculated.

FALSE

A notebook window is calculated.

rc (BOOL)

Success indicator.

TRUE

Coordinates were successfully calculated.

FALSE

Unable to calculate coordinates. This is returned if an invalid RECTL structure is specified in the *pRectl* parameter.

BKM_CALCPAGERECT - Syntax

This message calculates an application page rectangle from a notebook rectangle or calculates a notebook rectangle from an application page rectangle, depending on the setting of the *bPage* parameter.

```
param1
    RECTL  pRectl  /* Pointer to the RECTL structure that contains the coordinates of the rectangle. */

param2
    BOOL   bPage   /* Window specifier. */
```

BKM_CALCPAGERECT - Remarks

The application can use this message to determine the size of either the notebook window or the application page window. It can also be used when the application handles the position and size of the application page window.

To calculate the application page rectangle, specify the coordinates of the notebook window in the *pRectl* parameter and TRUE in the *bPage* parameter. The notebook control then uses the coordinates specified in the *pRectl* parameter to calculate and return the coordinates of the application page window.

To calculate the notebook rectangle, specify the coordinates of the application page window in the *pRectl* parameter and FALSE in the

bPage parameter. The notebook control then uses the coordinates specified in the *pRect* parameter to calculate and return the coordinates of the notebook window.

BKM_CALCPAGERECT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

BKM_CALCPAGERECT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

BKM_DELETEPAGE

BKM_DELETEPAGE Field - ulPageId

ulPageId ([ULONG](#))
Page identifier.

Page identifier for deletion. This is ignored if the BKA_ALL attribute of the *usDeleteFlag* parameter is specified.

BKM_DELETEPAGE Field - usDeleteFlag

usDeleteFlag ([USHORT](#))
Page range attribute.

Attribute that specifies the range of pages to be deleted.

BKA_SINGLE
Delete a single page.

BKA_TAB
If the page ID specified is that of a page with a major tab attribute, delete that page and all subsequent pages up to the next page that has a major tab attribute.

If the page ID specified is that of a page with a minor tab attribute, delete that page and all subsequent pages up to the next page that has either a major or minor tab attribute.

This attribute should only be specified for pages that have major or minor tab attributes. If a page with neither of these attributes is specified, FALSE is returned and no pages are deleted.

BKA_ALL

Delete all pages in the notebook.

BKM_DELETEPAGE Return Value - rc

rc (BOOL)

Success indicator.

TRUE

Pages were successfully deleted.

FALSE

Unable to delete the page or pages. This is returned if an invalid page ID is specified for the *ulPageId* parameter or if the BKA_TAB attribute is specified for a page that has neither a major nor a minor tab attribute.

BKM_DELETEPAGE - Parameters

ulPageId (ULONG)

Page identifier.

Page identifier for deletion. This is ignored if the BKA_ALL attribute of the *usDeleteFlag* parameter is specified.

usDeleteFlag (USHORT)

Page range attribute.

Attribute that specifies the range of pages to be deleted.

BKA_SINGLE

Delete a single page.

BKA_TAB

If the page ID specified is that of a page with a major tab attribute, delete that page and all subsequent pages up to the next page that has a major tab attribute.

If the page ID specified is that of a page with a minor tab attribute, delete that page and all subsequent pages up to the next page that has either a major or minor tab attribute.

This attribute should only be specified for pages that have major or minor tab attributes. If a page with neither of these attributes is specified, FALSE is returned and no pages are deleted.

BKA_ALL

Delete all pages in the notebook.

rc (BOOL)

Success indicator.

TRUE

Pages were successfully deleted.

FALSE

Unable to delete the page or pages. This is returned if an invalid page ID is specified for the *ulPageId* parameter or if the BKA_TAB attribute is specified for a page that has neither a major nor a minor tab attribute.

BKM_DELETEPAGE - Syntax

This message deletes the specified page or pages from the notebook data list.

```
param1
    ULONG    ulPageId      /* Page identifier. */

param2
    USHORT   usDeleteFlag  /* Page range attribute. */
```

BKM_DELETEPAGE - Remarks

The notebook frees all storage that it has allocated for the deleted page or pages. The application is responsible for deleting the application page window and bit map, if created.

BKM_DELETEPAGE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

BKM_DELETEPAGE - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

BKM_INSERTPAGE

BKM_INSERTPAGE Field - ulPageId

ulPageId (ULONG)
Page ID for placement.

Page identifier used for the placement of the inserted page. This identifier is ignored if the BKA_FIRST or BKA_LAST attribute of the *usPageOrder* parameter is specified.

BKM_INSERTPAGE Field - usPageStyle

usPageStyle (USHORT)
Style attributes.

Attributes that specify the style to be used for an inserted page. You can specify one attribute from each of the following groups by using logical OR operators (|) to combine attributes.

- Specify the following for automatic page position and size:

BKA_AUTOPAGESIZE
Notebook handles the positioning and sizing of the application page window specified in the [BKM_SETPAGEWINDOWHWND](#) message.
- Specify the following to display status area text:

BKA_STATUSTEXTON
Page is to be displayed with status area text. If this attribute is not specified, the application cannot associate a text string with the status area of the page being inserted.
- Specify one of the following if the page is to have a major or minor tab attribute:

BKA_MAJOR
Inserted page will have a major tab attribute.
BKA_MINOR
Inserted page will have a minor tab attribute.

BKM_INSERTPAGE Field - usPageOrder

usPageOrder (USHORT)
Order attributes.

Placement of page relative to the previously inserted pages. You can specify one of the following attributes:

- BKA_FIRST

Insert page at the front of the notebook. The page ID specified in the *ulPageId* parameter for *param1* is ignored if this is specified.
- BKA_LAST

Insert page at the end of the notebook. The page ID specified in the *ulPageId* parameter for *param1* is ignored if this is specified.
- BKA_NEXT

Insert page after the page whose ID is specified in the *ulPageId* parameter for *param1*. If the page ID specified in the *ulPageId* parameter is invalid, NULL is returned and no page is inserted.
- BKA_PREV

Insert page before the page whose ID is specified in the *ulPageId* parameter for *param1*. If the page ID specified in the *ulPageId* parameter is invalid, NULL is returned and no page is inserted.

BKM_INSERTPAGE Return Value - ulPageId

ulPageId (ULONG)

Page ID for insertion.

Identifier for the inserted page.

NULL

The page was not inserted into the notebook. An invalid page ID was specified for the *ulPageId* parameter for *param1* or not enough space was available to allocate the page data.

Other

Identifier for the inserted page.

BKM_INSERTPAGE - Parameters

ulPageId (ULONG)

Page ID for placement.

Page identifier used for the placement of the inserted page. This identifier is ignored if the BKA_FIRST or BKA_LAST attribute of the *usPageOrder* parameter is specified.

usPageStyle (USHORT)

Style attributes.

Attributes that specify the style to be used for an inserted page. You can specify one attribute from each of the following groups by using logical OR operators (|) to combine attributes.

- Specify the following for automatic page position and size:

BKA_AUTOPAGESIZE

Notebook handles the positioning and sizing of the application page window specified in the [BKM_SETPAGEWINDOWHWND](#) message.

- Specify the following to display status area text:

BKA_STATUSTEXTON

Page is to be displayed with status area text. If this attribute is not specified, the application cannot associate a text string with the status area of the page being inserted.

- Specify one of the following if the page is to have a major or minor tab attribute:

BKA_MAJOR
BKA_MINOR

Inserted page will have a major tab attribute.
Inserted page will have a minor tab attribute.

usPageOrder (USHORT)

Order attributes.

Placement of page relative to the previously inserted pages. You can specify one of the following attributes:

BKA_FIRST

Insert page at the front of the notebook. The page ID specified in the *ulPageId* parameter for *param1* is ignored if this is specified.

BKA_LAST

Insert page at the end of the notebook. The page ID specified in the *ulPageId* parameter for *param1* is ignored if this is specified.

BKA_NEXT

Insert page after the page whose ID is specified in the *ulPageId* parameter for *param1*. If the page ID specified in the *ulPageId* parameter is invalid, NULL is returned and no page is inserted.

BKA_PREV

Insert page before the page whose ID is specified in the *ulPageId* parameter for *param1*. If the page ID specified

in the *ulPageId* parameter is invalid, NULL is returned and no page is inserted.

ulPageId ([ULONG](#))

Page ID for insertion.

Identifier for the inserted page.

NULL

The page was not inserted into the notebook. An invalid page ID was specified for the *ulPageId* parameter for *param1* or not enough space was available to allocate the page data.

Other

Identifier for the inserted page.

BKM_INSERTPAGE - Syntax

This message inserts the specified page into the notebook data list.

```
param1
    ULONG    ulPageId    /* Page ID for placement. */

param2
    USHORT   usPageStyle /* Style attributes. */
    USHORT   usPageOrder /* Order attributes. */
```

BKM_INSERTPAGE - Remarks

The notebook control allocates and manages the storage needed for the new page. If neither the BKA_MAJOR or BKA_MINOR attribute is specified, the page is inserted with no tab attributes.

If the application does not specify the BKA_AUTOPAGESIZE attribute, it must handle the positioning and sizing of the application page window when it receives the BKN_NEWPAGESIZE notification code.

BKM_INSERTPAGE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

BKM_INSERTPAGE - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

BKM_INVALIDATETABS

BKM_INVALIDATETABS Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

BKM_INVALIDATETABS Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

BKM_INVALIDATETABS Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Tabs painted successfully.
FALSE	Tabs were not painted.

BKM_INVALIDATETABS - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

rc (BOOL)
Success indicator.

TRUE	Tabs painted successfully.
------	----------------------------

FALSE

Tabs were not painted.

BKM_INVALIDATETABS - Syntax

This message repaints all of the tabs in the notebook.

```
param1
    ULONG  ulReserved /* Reserved value, should be 0. */
param2
    ULONG  ulReserved /* Reserved value, should be 0. */
```

BKM_INVALIDATETABS - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

BKM_INVALIDATETABS - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Glossary](#)

BKM_QUERYPAGECOUNT

BKM_QUERYPAGECOUNT Field - ulPageId

ulPageId (ULONG)
Page ID or 0.

Page identifier from which to start the query, or 0. If this parameter is set to 0, the query begins with the first page.

BKM_QUERYPAGECOUNT Field - usQueryEnd

usQueryEnd (USHORT)

Query end attribute.

Attribute that ends the page count query.

BKA_MAJOR

Query the number of pages between the page ID specified in the *ulPageId* parameter and the next page that has the BKA_MAJOR attribute. The page that has the BKA_MAJOR attribute is not included in the page count.

BKA_MINOR

Query the number of pages between the page ID specified in the *ulPageId* parameter and the next page that has the BKA_MINOR attribute. The page that has the BKA_MINOR attribute is not included in the page count.

BKA_END

Query the number of pages between the page ID specified in the *ulPageId* parameter and the last page. When this attribute is specified, the page count includes the last page plus the notebook's back cover.

BKM_QUERYPAGECOUNT Return Value - pageCount

pageCount (SHORT)

Number of pages.

Number of pages in the notebook.

BOOKERR_INVALID_PARAMETERS

An invalid page ID was specified for the *ulPageId* parameter.

Other

Number of pages for the specified range. If the notebook is empty or no pages are found in the range, this value is 0.

BKM_QUERYPAGECOUNT - Parameters

ulPageId (ULONG)

Page ID or 0.

Page identifier from which to start the query, or 0. If this parameter is set to 0, the query begins with the first page.

usQueryEnd (USHORT)

Query end attribute.

Attribute that ends the page count query.

BKA_MAJOR

Query the number of pages between the page ID specified in the *ulPageId* parameter and the next page that has the BKA_MAJOR attribute. The page that has the BKA_MAJOR attribute is not included in the page count.

BKA_MINOR

Query the number of pages between the page ID specified in the *ulPageId* parameter and the next page that has the BKA_MINOR attribute. The page that has the BKA_MINOR attribute is not included in the page count.

BAK_END	Query the number of pages between the page ID specified in the <i>ulPageId</i> parameter and the last page. When this attribute is specified, the page count includes the last page plus the notebook's back cover.
pageCount (SHORT)	Number of pages.
	Number of pages in the notebook.
BOOKERR_INVALID_PARAMETERS	An invalid page ID was specified for the <i>ulPageId</i> parameter.
Other	Number of pages for the specified range. If the notebook is empty or no pages are found in the range, this value is 0.

BKM_QUERYPAGECOUNT - Syntax

This message queries the number of pages.

```
param1
    ULONG    ulPageId    /* Page ID or 0. */

param2
    USHORT   usQueryEnd  /* Query end attribute. */
```

BKM_QUERYPAGECOUNT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

BKM_QUERYPAGECOUNT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Default Processing](#)
 - [Glossary](#)

BKM_QUERYPAGECOUNT - BKM_QUERYPAGECOUNT

BKM_QUERYPAGEDATA Field - ulPageId

ulPageId (ULONG)
Page ID.

The page identifier of the page from which to retrieve the 4 bytes of data.

BKM_QUERYPAGEDATA Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

BKM_QUERYPAGEDATA Return Value - ulPageData

ulPageData (ULONG)
Page data.

Application-defined page data.

BOOKERR_INVALID_PARAMETERS
An invalid page ID was specified for the *ulPageId* parameter.

0
No page data was set for the page specified in the *ulPageId* parameter.

Other
Application-defined page data.

BKM_QUERYPAGEDATA - Parameters

ulPageId (ULONG)
Page ID.

The page identifier of the page from which to retrieve the 4 bytes of data.

ulReserved (ULONG)
Reserved value, should be 0.

ulPageData (ULONG)
Page data.

Application-defined page data.

BOOKERR_INVALID_PARAMETERS
An invalid page ID was specified for the *ulPageId* parameter.

0

	No page data was set for the page specified in the <i>ulPageId</i> parameter.
Other	Application-defined page data.

BKM_QUERYPAGEDATA - Syntax

This message queries the 4 bytes of application reserved storage associated with the specified page.

```
param1
    ULONG  ulPageId      /*  Page ID.  */

param2
    ULONG  ulReserved    /*  Reserved value, should be 0.  */
```

BKM_QUERYPAGEDATA - Remarks

This data is set by using the [BKM_SETPAGEDATA](#) message.

BKM_QUERYPAGEDATA - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

BKM_QUERYPAGEDATA - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

BKM_QUERYPAGEID

BKM_QUERYPAGEID Field - ulPageId

ulPageId (ULONG)
Location page ID.

Page identifier used for locating the requested page. This identifier is ignored if the BKA_FIRST, BKA_LAST, or BKA_TOP attribute is specified.

BKM_QUERYPAGEID Field - usQueryOrder

usQueryOrder (USHORT)
Page ID query order.

Order in which to query the page identifier.

- BKA_FIRST
Get the page identifier for the first page. The page ID specified in the *ulPageId* parameter for *param1* is ignored if this is specified.
- BKA_LAST
Get the page identifier for the last page. The page ID specified in the *ulPageId* parameter for *param1* is ignored if this is specified.
- BKA_NEXT
Get the page identifier for the page after the page whose ID is specified in the *ulPageId* parameter for *param1*. If the page ID specified in the *ulPageId* parameter is invalid, BOOKERR_INVALID_PARAMETERS is returned.
- BKA_PREV
Get the page identifier for the page before the page whose ID is specified in the *ulPageId* parameter for *param1*. If the page ID specified in the *ulPageId* parameter is invalid, BOOKERR_INVALID_PARAMETERS is returned.
- BKA_TOP
Get the page identifier for the page currently visible in the notebook. The page ID specified in the *ulPageId* parameter for *param1* is ignored if this is specified.

BKM_QUERYPAGEID Field - usPageStyle

usPageStyle (USHORT)
Page style.

Page style for which to query the page identifier. If neither of these attributes is specified, the *usPageStyle* parameter is ignored.

- BKA_MAJOR
Query page with major tab attribute.
- BKA_MINOR
Query page with minor tab attribute. If a major tab page is found before the minor tab page, the search is ended and 0 is returned.

BKM_QUERYPAGEID Return Value - ulPageId

ulPageId (ULONG)

Retrieved page ID.

BOOKERR_INVALID_PARAMETERS

Returned if the page ID specified for the *ulPageId* parameter for *param1* is invalid when specifying either the BKA_PREV or BKA_NEXT attribute in the *usQueryOrder* parameter.

0

Requested page not found. This could be an indication that the end or front of the list has been reached, or that the notebook is empty.

Other

Retrieved page identifier.

BKM_QUERYPAGEID - Parameters

ulPageId (ULONG)

Location page ID.

Page identifier used for locating the requested page. This identifier is ignored if the BKA_FIRST, BKA_LAST, or BKA_TOP attribute is specified.

usQueryOrder (USHORT)

Page ID query order.

Order in which to query the page identifier.

BKA_FIRST

Get the page identifier for the first page. The page ID specified in the *ulPageId* parameter for *param1* is ignored if this is specified.

BKA_LAST

Get the page identifier for the last page. The page ID specified in the *ulPageId* parameter for *param1* is ignored if this is specified.

BKA_NEXT

Get the page identifier for the page after the page whose ID is specified in the *ulPageId* parameter for *param1*. If the page ID specified in the *ulPageId* parameter is invalid, BOOKERR_INVALID_PARAMETERS is returned.

BKA_PREV

Get the page identifier for the page before the page whose ID is specified in the *ulPageId* parameter for *param1*. If the page ID specified in the *ulPageId* parameter is invalid, BOOKERR_INVALID_PARAMETERS is returned.

BKA_TOP

Get the page identifier for the page currently visible in the notebook. The page ID specified in the *ulPageId* parameter for *param1* is ignored if this is specified.

usPageStyle (USHORT)

Page style.

Page style for which to query the page identifier. If neither of these attributes is specified, the *usPageStyle* parameter is ignored.

BKA_MAJOR

Query page with major tab attribute.

BKA_MINOR

Query page with minor tab attribute. If a major tab page is found before the minor tab page, the search is ended and 0 is returned.

ulPageId (ULONG)

Retrieved page ID.

BOOKERR_INVALID_PARAMETERS

Returned if the page ID specified for the *ulPageId* parameter for *param1* is invalid when specifying either the BKA_PREV or BKA_NEXT attribute in the *usQueryOrder* parameter.

0	Requested page not found. This could be an indication that the end or front of the list has been reached, or that the notebook is empty.
Other	Retrieved page identifier.

BKM_QUERYPAGEID - Syntax

This message queries the page identifier for the specified page.

```
param1
    ULONG    ulPageId      /* Location page ID. */

param2
    USHORT    usQueryOrder /* Page ID query order. */
    USHORT    usPageStyle  /* Page style. */
```

BKM_QUERYPAGEID - Remarks

If the BKA_FIRST, BKA_LAST, or BKA_TOP attribute is specified, the page ID in the *ulPageId* parameter is ignored.

BKM_QUERYPAGEID - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

BKM_QUERYPAGEID - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

BKM_QUERYPAGEINFO

BKM_QUERYPAGEINFO Field - ulPageId

ulPageId (ULONG)
Id of the notebook page whose information is to be queried.

BKM_QUERYPAGEINFO Field - pPageInfo

pPageInfo (PPAGEINFO)
Pointer to a notebook page information structure.

BKM_QUERYPAGEINFO Field - rc

rc (BOOL)
Success indicator.

Possible values are described in the following list:

TRUE	Message was processed.
FALSE	Message was ignored.

BKM_QUERYPAGEINFO - Parameters

ulPageId (ULONG)
Id of the notebook page whose information is to be queried.

pPageInfo (PPAGEINFO)
Pointer to a notebook page information structure.

rc (BOOL)
Success indicator.

Possible values are described in the following list:

TRUE	Message was processed.
FALSE	Message was ignored.

BKM_QUERYPAGEINFO - Syntax

This message queries the page information associated with a notebook page.

```
param1
    ULONG      ulPageId    /* Id of the notebook page whose information is to be queried.

param2
    PPAGEINFO  pPageInfo  /* Pointer to a notebook page information structure.

returns
    BOOL      rc          /* Success indicator. */
```

BKM_QUERYPAGEINFO - Remarks

This message handles the following notebook messages:

- [BKM_QUERYPAGEDATA](#)
- [BKM_QUERYPAGEWINDOWHWND](#)
- [BKM_QUERYSTATUSLINETEXT](#)
- [BKM_QUERYTABBITMAP](#)
- [BKM_QUERYTABTEXT](#)

BKM_QUERYPAGEINFO - Default Processing

The default message procedure sets *rc* to TRUE.

BKM_QUERYPAGEINFO - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

BKM_QUERYPAGESTYLE

BKM_QUERYPAGESTYLE Field - ulPageId

ulPageId (ULONG)
Page ID.

Page identifier of the page from which to query the style setting.

BKM_QUERYPAGESTYLE Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

BKM_QUERYPAGESTYLE Return Value - usPageStyle

usPageStyle (USHORT)
Page style data.

BOOKERR_INVALID_PARAMETERS
An invalid page ID was specified for the *ulPageId* parameter.

Other
Page style data.

BKM_QUERYPAGESTYLE - Parameters

ulPageId (ULONG)
Page ID.

Page identifier of the page from which to query the style setting.

ulReserved (ULONG)
Reserved value, should be 0.

usPageStyle (USHORT)
Page style data.

BOOKERR_INVALID_PARAMETERS
An invalid page ID was specified for the *ulPageId* parameter.

Other
Page style data.

BKM_QUERYPAGESTYLE - Syntax

This message queries the style that was set when the specified page was inserted.

```
param1
    ULONG    ulPageId    /* Page ID. */

param2
    ULONG    ulReserved  /* Reserved value, should be 0. */
```

BKM_QUERYPAGESTYLE - Remarks

This style data is set when the page is inserted, which is done by using the [BKM_INSERTPAGE](#) message.

BKM_QUERYPAGESTYLE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

BKM_QUERYPAGESTYLE - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

BKM_QUERYPAGEWINDOWHWND

BKM_QUERYPAGEWINDOWHWND Field - ulPageId

ulPageId ([ULONG](#))
Page ID.

Page identifier of the page whose window handle is requested.

BKM_QUERYPAGEWINDOWHWND Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

BKM_QUERYPAGEWINDOWHWND Return Value - hwndPage

hwndPage (HWND)
Window handle.

Handle of the application page window associated with the specified page identifier.

BOOKERR_INVALID_PARAMETERS
An invalid page ID was specified for the *ulPageId* parameter.

NULLHANDLE
No application page window handle is associated for the page specified in the *ulPageId* parameter.

Other
Handle of the application page window associated with the specified page identifier.

BKM_QUERYPAGEWINDOWHWND - Parameters

ulPageId (ULONG)
Page ID.

Page identifier of the page whose window handle is requested.

ulReserved (ULONG)
Reserved value, should be 0.

hwndPage (HWND)
Window handle.

Handle of the application page window associated with the specified page identifier.

BOOKERR_INVALID_PARAMETERS
An invalid page ID was specified for the *ulPageId* parameter.

NULLHANDLE
No application page window handle is associated for the page specified in the *ulPageId* parameter.

Other
Handle of the application page window associated with the specified page identifier.

BKM_QUERYPAGEWINDOWHWND - Syntax

This message queries the application page window handle associated with the specified page.

```
param1
    ULONG    ulPageId    /* Page ID. */

param2
    ULONG    ulReserved  /* Reserved value, should be 0. */
```

BKM_QUERYPAGEWINDOWHWND - Remarks

The application page window handle is set by using the [BKM_SETPAGEWINDOWHWND](#) message.

BKM_QUERYPAGEWINDOWHWND - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return `NULLHANDLE`.

BKM_QUERYPAGEWINDOWHWND - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

BKM_QUERYSTATUSLINETEXT

BKM_QUERYSTATUSLINETEXT Field - ulPageId

ulPageId ([ULONG](#))
Page ID.

Page identifier of the page whose status line text is requested.

BKM_QUERYSTATUSLINETEXT Field - pBookText

pBookText ([PBOOKTEXT](#))
Pointer to a [BOOKTEXT](#) data structure.

BKM_QUERYSTATUSLINETEXT Return Value - statusTextLen

statusTextLen ([USHORT](#))
String length.

Length of the status line text string.

BOOKERR_INVALID_PARAMETERS
An invalid page ID was specified for the *ulPageId* parameter or the structure specified for the *pBookText* parameter is invalid.

0
No text data has been set ([BKM_SETSTATUSLINETEXT](#)) for the page specified in the *ulPageId* parameter.

Other
Length of the returned status line text string.

BKM_QUERYSTATUSLINETEXT - Parameters

ulPageId ([ULONG](#))
Page ID.

Page identifier of the page whose status line text is requested.

pBookText ([PBOOKTEXT](#))
Pointer to a [BOOKTEXT](#) data structure.

statusTextLen ([USHORT](#))
String length.

Length of the status line text string.

BOOKERR_INVALID_PARAMETERS
An invalid page ID was specified for the *ulPageId* parameter or the structure specified for the *pBookText* parameter is invalid.

0
No text data has been set ([BKM_SETSTATUSLINETEXT](#)) for the page specified in the *ulPageId* parameter.

Other
Length of the returned status line text string.

BKM_QUERYSTATUSLINETEXT - Syntax

This message queries the status line text, text size, or both for the specified page.

```
param1
    ULONG        ulPageId        /* Page ID. */

param2
    PBOOKTEXT    pBookText       /* Pointer to a BOOKTEXT data structure. */
```

BKM_QUERYSTATUSLINETEXT - Remarks

The size of the status line text string can be queried by specifying 0 for the *textLen* field of the [BOOKTEXT](#) data structure. In this way, the application can determine the size of the buffer needed to store the status line text string. The null character at the end of the text string is not included in the returned length.

BKM_QUERYSTATUSLINETEXT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action other than to return 0.

BKM_QUERYSTATUSLINETEXT - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

BKM_QUERYTABBITMAP

BKM_QUERYTABBITMAP Field - ulPageId

ulPageId ([ULONG](#))
Page ID.

Page identifier of the page whose bit-map handle is requested. This should be a page for which a BKA_MAJOR or BKA_MINOR attribute has been specified.

BKM_QUERYTABBITMAP Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

BKM_QUERYTABBITMAP Return Value - hbm

hbm (HBITMAP)
Bit-map handle.

Handle of the bit map associated with the specified page identifier.

BOOKERR_INVALID_PARAMETERS
An invalid page ID was specified for the *ulPageId* parameter.

NULLHANDLE
No bit-map handle is associated with the page specified in the *ulPageId* parameter.

Other
Handle of the bit map associated with the specified page identifier.

BKM_QUERYTABBITMAP - Parameters

ulPageId (ULONG)
Page ID.

Page identifier of the page whose bit-map handle is requested. This should be a page for which a BKA_MAJOR or BKA_MINOR attribute has been specified.

ulReserved (ULONG)
Reserved value, should be 0.

hbm (HBITMAP)
Bit-map handle.

Handle of the bit map associated with the specified page identifier.

BOOKERR_INVALID_PARAMETERS
An invalid page ID was specified for the *ulPageId* parameter.

NULLHANDLE
No bit-map handle is associated with the page specified in the *ulPageId* parameter.

Other
Handle of the bit map associated with the specified page identifier.

BKM_QUERYTABBITMAP - Syntax

This message queries the bit-map handle associated with the specified page.

```
param1
    ULONG    ulPageId    /* Page ID. */

param2
    ULONG    ulReserved  /* Reserved value, should be 0. */
```

BKM_QUERYTABBITMAP - Remarks

The tab bit-map handle is set by using the [BKM_SETTABBITMAP](#) message.

If this message is sent for a page having both major and minor tab attributes, the notebook returns the bit map associated with the major tab.

BKM_QUERYTABBITMAP - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return `NULLHANDLE`.

BKM_QUERYTABBITMAP - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

BKM_QUERYTABTEXT

BKM_QUERYTABTEXT Field - ulPageId

ulPageId ([ULONG](#))
Page ID.

Page identifier of the page whose tab text is requested. This should be a page for which a BKA_MAJOR or BKA_MINOR attribute has been specified.

BKM_QUERYTABTEXT Field - pBookText

pBookText (PBOOKTEXT)
Pointer to a BOOKTEXT data structure.

BKM_QUERYTABTEXT Return Value - tabTextLen

tabTextLen (USHORT)
Length of the tab text string.

BOOKERR_INVALID_PARAMETERS
An invalid page ID was specified for the *ulPageId* parameter or the structure specified for the *pBookText* parameter is invalid.

0
No text data has been set (BKM_SETTABTEXT) for the page specified in the *ulPageId* parameter.

Other
Length of the returned tab text string.

BKM_QUERYTABTEXT - Parameters

ulPageId (ULONG)
Page ID.

Page identifier of the page whose tab text is requested. This should be a page for which a BKA_MAJOR or BKA_MINOR attribute has been specified.

pBookText (PBOOKTEXT)
Pointer to a BOOKTEXT data structure.

tabTextLen (USHORT)
Length of the tab text string.

BOOKERR_INVALID_PARAMETERS
An invalid page ID was specified for the *ulPageId* parameter or the structure specified for the *pBookText* parameter is invalid.

0
No text data has been set (BKM_SETTABTEXT) for the page specified in the *ulPageId* parameter.

Other
Length of the returned tab text string.

BKM_QUERYTABTEXT - Syntax

This message queries the text, text size, or both for the specified page.

```
param1
    ULONG      ulPageId    /* Page ID. */

param2
    BOOKTEXT   pBookText   /* Pointer to a BOOKTEXT data structure. */
```

BKM_QUERYTABTEXT - Remarks

The size of the tab text string can be queried by specifying 0 for the *tabTextLen* field in the [BOOKTEXT](#) data structure. In this way, the application can determine the size of the buffer needed to store the tab text string. The null character at the end of the text string is not included in the returned length.

If this message is sent for a page having both major and minor tab attributes, the notebook returns the text which is associated with the major tab.

BKM_QUERYTABTEXT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

BKM_QUERYTABTEXT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

BKM_SETDIMENSIONS

BKM_SETDIMENSIONS Field - usWidth

usWidth ([USHORT](#))
Width value to set.

BKM_SETDIMENSIONS Field - usHeight

usHeight ([USHORT](#))
Height value to set.

BKM_SETDIMENSIONS Field - usType

usType ([USHORT](#))
Notebook region.

Notebook region for which the dimensions are to be set. Valid values are:

- BKA_MAJORTAB
- BKA_MINORTAB
- BKA_PAGEBUTTON.

BKM_SETDIMENSIONS Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Dimensions were successfully set.
FALSE	Unable to set dimensions. Returned if an invalid value is specified for the <i>usType</i> parameter or if the dimensions are invalid.

BKM_SETDIMENSIONS - Parameters

usWidth ([USHORT](#))
Width value to set.

usHeight ([USHORT](#))
Height value to set.

usType ([USHORT](#))
Notebook region.

Notebook region for which the dimensions are to be set. Valid values are:

- BKA_MAJORTAB
- BKA_MINORTAB
- BKA_PAGEBUTTON.

rc ([BOOL](#))

Success indicator.

TRUE

Dimensions were successfully set.

FALSE

Unable to set dimensions. Returned if an invalid value is specified for the *usType* parameter or if the dimensions are invalid.

BKM_SETDIMENSIONS - Syntax

This message sets the height and width for the major tabs, minor tabs, or page buttons.

```
param1
    USHORT  usWidth  /* Width value to set. */
    USHORT  usHeight /* Height value to set. */

param2
    USHORT  usType    /* Notebook region. */
```

BKM_SETDIMENSIONS - Remarks

If either the BKA_MAJORTAB or BKA_MINORTAB attribute is specified for the *usType* parameter, the minimum width and height for display is 7 pels to allow space for the tab border and the selection cursor. If the tabs or page buttons are not to be displayed, the height and width can be set to 0.

If the new dimensions cause the notebook size to change, the notebook sends a BKN_NEWPAGESIZE notification code to the application.

BKM_SETDIMENSIONS - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

BKM_SETDIMENSIONS - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

BKM_SETNOTEBOOKBUTTONS

BKM_SETNOTEBOOKBUTTONS Field - ulCount

ulCount ([ULONG](#))
Number of buttons to set.

BKM_SETNOTEBOOKBUTTONS Field - pButtonArray

pButtonArray ([PNOTEBOOKBUTTON](#))
Notebook button array.

Array of NOTEBOOKBUTTON structures that specify the style and content of the buttons in the common button area. The *ulCount* parameter indicates the number of entries in the array.

BKM_SETNOTEBOOKBUTTONS Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Buttons were successfully set.
FALSE	Unable to set buttons.

BKM_SETNOTEBOOKBUTTONS - Parameters

ulCount ([ULONG](#))
Number of buttons to set.

pButtonArray ([PNOTEBOOKBUTTON](#))
Notebook button array.

Array of NOTEBOOKBUTTON structures that specify the style and content of the buttons in the common button area. The *ulCount* parameter indicates the number of entries in the array.

rc ([BOOL](#))
Success indicator.

TRUE	Buttons were successfully set.
FALSE	Unable to set buttons.

BKM_SETNOTEBOOKBUTTONS - Syntax

This message creates a set of buttons in the common area of the notebook page,

```
param1
    ULONG          ulCount      /* Number of buttons to set. */

param2
    PNOTEBOOKBUTTON pButtonArray /* Notebook button array. */
```

BKM_SETNOTEBOOKBUTTONS - Remarks

This message works only for notebooks with the styles BKS_TABBEDDIALOG and BKS_BUTTONAREA.

All notebook buttons are created as children of the notebook window and must have an ID of less than BKA_MAXBUTTONID. An application can send messages to the notebook buttons but is also responsible for making sure that they are in the expected state when a new page is turned to.

The notebook buttons created using this message will be displayed for all pages in the notebook, except for pages containing other push buttons with the BS_NOTEBOOKBUTTON style. WM_COMMAND, WM_SYSCOMMAND, and WM_HELP messages from these buttons are passed on to the top page window procedure for processing.

The notebook is responsible for destroying buttons created by this message.

BKM_SETNOTEBOOKBUTTONS - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

BKM_SETNOTEBOOKBUTTONS - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

BKM_SETNOTEBOOKCOLORS

BKM_SETNOTEBOOKCOLORS Field - ulColor

ulColor (ULONG)
Color value to set.

BKM_SETNOTEBOOKCOLORS Field - usBookAttr

usBookAttr (USHORT)
Notebook region.

Notebook region whose color is to be set. Valid values are:

 BKA_BACKGROUNDPAGECOLOR or BKA_BACKGROUNDPAGECOLORINDEX
 Page background. This color is initially set to SYSCLR_PAGEBACKGROUND.

 BKA_BACKGROUNDMAJORCOLOR or BKA_BACKGROUNDMAJORCOLORINDEX
 Major tab background. This color is initially set to SYSCLR_PAGEBACKGROUND.

 BKA_BACKGROUNDMINORCOLOR or BKA_BACKGROUNDMINORCOLORINDEX
 Minor tab background. This color is initially set to SYSCLR_PAGEBACKGROUND.

 BKA_FOREGROUNDMAJORCOLOR or BKA_FOREGROUNDMAJORCOLORINDEX
 Major tab text. This color is initially set to SYSCLR_WINDOWTEXT.

 BKA_FOREGROUNDMINORCOLOR or BKA_FOREGROUNDMINORCOLORINDEX
 Minor tab text. This color is initially set to SYSCLR_WINDOWTEXT.

BKM_SETNOTEBOOKCOLORS Return Value - rc

rc (BOOL)
Success indicator.

 TRUE
 Colors were successfully set.

 FALSE
 Unable to set colors. Returned if an invalid notebook attribute is specified for the *usBookAttr* parameter.

BKM_SETNOTEBOOKCOLORS - Parameters

ulColor ([ULONG](#))
Color value to set.

usBookAttr ([USHORT](#))
Notebook region.

Notebook region whose color is to be set. Valid values are:

BJA_BACKGROUNDPAGECOLOR or **BJA_BACKGROUNDPAGECOLORINDEX**
 Page background. This color is initially set to **SYSCLR_PAGEBACKGROUND**.

BJA_BACKGROUNDMAJORCOLOR or **BJA_BACKGROUNDMAJORCOLORINDEX**
 Major tab background. This color is initially set to **SYSCLR_PAGEBACKGROUND**.

BJA_BACKGROUNDMINORCOLOR or **BJA_BACKGROUNDMINORCOLORINDEX**
 Minor tab background. This color is initially set to **SYSCLR_PAGEBACKGROUND**.

BJA_FOREGROUNDMAJORCOLOR or **BJA_FOREGROUNDMAJORCOLORINDEX**
 Major tab text. This color is initially set to **SYSCLR_WINDOWTEXT**.

BJA_FOREGROUNDMINORCOLOR or **BJA_FOREGROUNDMINORCOLORINDEX**
 Minor tab text. This color is initially set to **SYSCLR_WINDOWTEXT**.

rc ([BOOL](#))
Success indicator.

TRUE
 Colors were successfully set.

FALSE
 Unable to set colors. Returned if an invalid notebook attribute is specified for the *usBookAttr* parameter.

BKM_SETNOTEBOOKCOLORS - Syntax

This message sets the colors for the major tab text and background, the minor tab text and background, and the notebook page background.

```
param1
    ULONG    ulColor      /* Color value to set. */

param2
    USHORT   usBookAttr   /* Notebook region. */
```

BKM_SETNOTEBOOKCOLORS - Remarks

The notebook background, border, selection cursor, and status line text colors are mapped to system presentation attributes. See [WM_PRESPARAMCHANGED](#) (in [Notebook Controls](#)) for information about setting the color of these regions.

BKM_SETNOTEBOOKCOLORS - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

BKM_SETNOTEBOOKCOLORS - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

BKM_SETPAGEDATA

BKM_SETPAGEDATA Field - ulPageId

ulPageId ([ULONG](#))
Page ID.
The page identifier of the page from which to set the 4 bytes of data.

BKM_SETPAGEDATA Field - ulPageData

ulPageData ([ULONG](#))
Page data.
Application-defined page data.

BKM_SETPAGEDATA Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Page data was successfully set.
FALSE	Unable to set page data. This value is returned if the page ID specified in the <i>ulPageId</i> parameter is invalid.

BKM_SETPAGEDATA - Parameters

ulPageId ([ULONG](#))

Page ID.

The page identifier of the page from which to set the 4 bytes of data.

ulPageData ([ULONG](#))

Page data.

Application-defined page data.

rc ([BOOL](#))

Success indicator.

TRUE

Page data was successfully set.

FALSE

Unable to set page data. This value is returned if the page ID specified in the *ulPageId* parameter is invalid.

BKM_SETPAGEDATA - Syntax

This message sets the 4 bytes of application reserved storage associated with the specified page.

```
param1
    ULONG  ulPageId    /* Page ID. */

param2
    ULONG  ulPageData  /* Page data. */
```

BKM_SETPAGEDATA - Remarks

This data can be queried by using the [BKM_QUERYPAGEDATA](#) message.

BKM_SETPAGEDATA - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

BKM_SETPAGEDATA - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

BKM_SETPAGEINFO

BKM_SETPAGEINFO Field - ulPageld

ulPageld ([ULONG](#))
Id of the notebook page whose information is to be set.

BKM_SETPAGEINFO Field - pPageInfo

pPageInfo ([PPAGEINFO](#))
Pointer to a notebook page information structure.

BKM_SETPAGEINFO Field - rc

rc ([BOOL](#))
Success indicator.

Possible values are described in the following list:

TRUE	Message was processed.
FALSE	Message was ignored.

BKM_SETPAGEINFO - Parameters

ulPageId ([ULONG](#))

Id of the notebook page whose information is to be set.

pPageInfo ([PPAGEINFO](#))

Pointer to a notebook page information structure.

rc ([BOOL](#))

Success indicator.

Possible values are described in the following list:

TRUE

Message was processed.

FALSE

Message was ignored.

BKM_SETPAGEINFO - Syntax

This message sets the page information associated with notebook page which contains a single message.

```
param1
    ULONG        ulPageId    /* Id of the notebook page whose information is to be set. */

param2
    PPAGEINFO    pPageInfo   /* Pointer to a notebook page information structure. */

returns
    BOOL         rc          /* Success indicator. */
```

BKM_SETPAGEINFO - Remarks

This message provides an application with the ability to associate a window handle, a static dialog resource or a dynamic dialog resource with a notebook page. The notebook can automatically load the dialog resource when the resource is associated with the page or when the page is turned.

This message performs the tasks of the following notebook messages:

- [BKM_SETPAGEDATA](#)
- [BKM_SETPAGEWINDOWHWND](#)
- [BKM_SETSTATUSLINETEXT](#)
- [BKM_SETTABBITMAP](#)
- [BKM_SETTABTEXT](#)

BKM_SETPAGEINFO - Default Processing

The default message procedure sets *rc* to TRUE.

BKM_SETPAGEINFO - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

BKM_SETPAGEWINDOWHWND

BKM_SETPAGEWINDOWHWND Field - ulPageId

ulPageId ([ULONG](#))
Page ID.

The page ID of the notebook page with which the application page window is to be associated.

BKM_SETPAGEWINDOWHWND Field - hwndPage

hwndPage ([HWND](#))
Window handle.

The handle of the application page window that is to be associated with the notebook page identified in the *ulPageId* parameter.

BKM_SETPAGEWINDOWHWND Return Value - rc

rc ([BOOL](#))
Success indicator.

- | | |
|-------|---|
| TRUE | Application page window handle was successfully set. |
| FALSE | Unable to set application page window handle. This value is returned if the page ID specified for the <i>ulPageId</i> parameter is invalid. |
-

BKM_SETPAGEWINDOWHWND - Parameters

ulPageId ([ULONG](#))

Page ID.

The page ID of the notebook page with which the application page window is to be associated.

hwndPage ([HWND](#))

Window handle.

The handle of the application page window that is to be associated with the notebook page identified in the *ulPageId* parameter.

rc ([BOOL](#))

Success indicator.

TRUE

Application page window handle was successfully set.

FALSE

Unable to set application page window handle. This value is returned if the page ID specified for the *ulPageId* parameter is invalid.

BKM_SETPAGEWINDOWHWND - Syntax

This message associates an application page window handle with the specified notebook page.

```
param1
    ULONG   ulPageId /* Page ID. */

param2
    HWND    hwndPage /* Window handle. */
```

BKM_SETPAGEWINDOWHWND - Remarks

The notebook shows the application page window specified in the *hwndPage* parameter whenever the notebook page specified in the *ulPageId* parameter is brought to the top of the notebook. If the BKA_AUTOPAGESIZE attribute is specified when that page is inserted into the notebook, the notebook also handles the sizing and positioning of the application page window.

BKM_SETPAGEWINDOWHWND - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

BKM_SETPAGEWINDOWHWND - Topics

Select an item:

[Syntax](#)

BKM_SETSTATUSLINETEXT

BKM_SETSTATUSLINETEXT Field - ulPageld

ulPageld ([ULONG](#))
Page ID.

The page identifier with which to associate the text string.

BKM_SETSTATUSLINETEXT Field - pString

pString ([PSZ](#))
Pointer to a text string that ends in a null character.

BKM_SETSTATUSLINETEXT Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Status line text was successfully set.
FALSE	Unable to set status line text. This value is returned if the page ID specified in the <i>ulPageld</i> parameter is invalid or if the page was inserted without specifying the BKA_STATUSTEXTON attribute.

BKM_SETSTATUSLINETEXT - Parameters

ulPageld ([ULONG](#))
Page ID.

The page identifier with which to associate the text string.

pString ([PSZ](#))

Pointer to a text string that ends in a null character.

rc ([BOOL](#))

Success indicator.

TRUE

Status line text was successfully set.

FALSE

Unable to set status line text. This value is returned if the page ID specified in the *ulPageId* parameter is invalid or if the page was inserted without specifying the BKA_STATUSTEXTON attribute.

BKM_SETSTATUSLINETEXT - Syntax

This message associates a text string with the specified page's status line.

```
param1
    ULONG    ulPageId    /* Page ID. */

param2
    PSZ      pString     /* Pointer to a text string that ends in a null character. */
```

BKM_SETSTATUSLINETEXT - Remarks

If the text is longer than the status area length, only the text that fits in the status area is displayed.

BKM_SETSTATUSLINETEXT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

BKM_SETSTATUSLINETEXT - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

BKM_SETTABBITMAP

BKM_SETTABBITMAP Field - ulPageld

ulPageld (ULONG)
Page ID.

The page identifier with which to associate the bit-map handle. This should be a page for which a BKA_MAJOR or BKA_MINOR attribute has been specified.

BKM_SETTABBITMAP Field - hbm

hbm (HBITMAP)
Bit-map handle.

BKM_SETTABBITMAP Return Value - rc

rc (BOOL)
Success indicator.

TRUE
Tab bit map was successfully set.

FALSE
Unable to set tab bit map. If the page ID specified in the *ulPageld* parameter is invalid or if it identifies a page that does not have a BKA_MAJOR or BKA_MINOR attribute, FALSE is returned and no bit map is associated with the page.

BKM_SETTABBITMAP - Parameters

ulPageld (ULONG)
Page ID.

The page identifier with which to associate the bit-map handle. This should be a page for which a BKA_MAJOR or BKA_MINOR attribute has been specified.

hbm (HBITMAP)
Bit-map handle.

rc (BOOL)
Success indicator.

TRUE

Tab bit map was successfully set.

FALSE

Unable to set tab bit map. If the page ID specified in the *ulPageId* parameter is invalid or if it identifies a page that does not have a BKA_MAJOR or BKA_MINOR attribute, FALSE is returned and no bit map is associated with the page.

BKM_SETTABBITMAP - Syntax

This message associates a bit-map handle with the specified page.

```
param1
    ULONG    ulPageId /* Page ID. */

param2
    HBITMAP  hbm       /* Bit-map handle. */
```

BKM_SETTABBITMAP - Remarks

If this message is sent for a page having both major and minor tab attributes, the notebook sets both the major and minor tab bit maps.

When displayed, the bit map is stretched to fit the size of the tab. If a tab has rounded or polygonal edges, the bit map is sized to fit the rectangular area of the tab, as shown in the following picture.

Bit Map Stretched to Fit Rectangular Area



Square
Tab



Rounded
Tab



Polygonal
Tab

BKM_SETTABBITMAP - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

BKM_SETTABBITMAP - Topics

Select an item:

[Syntax](#)

[Parameters](#)

BKM_SETTABCOLOR

BKM_SETTABCOLOR Field - ulPageId

ulPageId ([ULONG](#))
Page ID.

Page identifier with which to associate the tab color. Specify zero to change all the tabs to a specific color. This parameter is ignored if the BKV_AUTOCOLOR attribute has been specified for the *ulPageId* parameter.

BKM_SETTABCOLOR Field - rgbColor

rgbColor ([ULONG](#))
Tab color

Color that the tab should be set to. Additionally, the following value may be used:

BKV_AUTOCOLOR Automatically paint the tabs with a default set of colors.

BKM_SETTABCOLOR Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE Tab color was successfully set.

FALSE Unable to set tab color. If the page ID specified in the *ulPageId* parameter is invalid, FALSE is returned and no action is taken.

BKM_SETTABCOLOR - Parameters

ulPageId (ULONG)

Page ID.

Page identifier with which to associate the tab color. Specify zero to change all the tabs to a specific color. This parameter is ignored if the BKV_AUTOCOLOR attribute has been specified for the *ulPageId* parameter.

rgbColor (ULONG)

Tab color

Color that the tab should be set to. Additionally, the following value may be used:

BKV_AUTOCOLOR

Automatically paint the tabs with a default set of colors.

rc (BOOL)

Success indicator.

TRUE

Tab color was successfully set.

FALSE

Unable to set tab color. If the page ID specified in the *ulPageId* parameter is invalid, FALSE is returned and no action is taken.

BKM_SETTABCOLOR - Syntax

This message can be sent to a notebook to set the background color of a tab.

```
param1
    ULONG  ulPageId /* Page ID. */

param2
    ULONG  rgbColor /* Tab color */
```

BKM_SETTABCOLOR - Remarks

This message works only for notebooks with the style BKS_TABBEDDIALOG. The colors set by BKV_AUTOTABCOLORS are as follows:
0x0055DBFF 0x0080DBAA 0x008092FF 0x00D5B6AA 0x00FFFFFFAA
0x00AA92AA 0x00FF9255 0x00FFDB55 0x00FFB6AA 0x00FFDBAA
In 16-color mode, or when the Palette Manager is active in 256-color mode, notebooks that are in auto-color mode switch to a set of dithered colors, making sure that the tab text is readable.

The application can change the color of the tab again by sending additional BKM_SETTABCOLOR messages for the specified page.

If the page ID specified in the *ulPageId* is zero, all the tabs are set to the specified color.

BKM_SETTABCOLOR - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

BKM_SETTABCOLOR - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

BKM_SETTABTEXT

BKM_SETTABTEXT Field - ulPageId

ulPageId ([ULONG](#))
Page ID.

The page identifier with which to associate the text string. This should be a page for which a BKA_MAJOR or BKA_MINOR attribute has been specified.

BKM_SETTABTEXT Field - pString

pString ([PSZ](#))
Pointer to a text string that ends with a null character.

BKM_SETTABTEXT Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Tab text was successfully set.
FALSE	Unable to set tab text. If the page ID specified in the <i>ulPageId</i> parameter is invalid or if it identifies a page that does not have a BKA_MAJOR or BKA_MINOR attribute, FALSE is returned and no text string is associated with the page.

BKM_SETTABTEXT - Parameters

ulPageId ([ULONG](#))

Page ID.

The page identifier with which to associate the text string. This should be a page for which a BKA_MAJOR or BKA_MINOR attribute has been specified.

pString ([PSZ](#))

Pointer to a text string that ends with a null character.

rc ([BOOL](#))

Success indicator.

TRUE

Tab text was successfully set.

FALSE

Unable to set tab text. If the page ID specified in the *ulPageId* parameter is invalid or if it identifies a page that does not have a BKA_MAJOR or BKA_MINOR attribute, FALSE is returned and no text string is associated with the page.

BKM_SETTABTEXT - Syntax

This message associates a text string with the specified page.

```
param1
    ULONG  ulPageId  /* Page ID. */

param2
    PSZ     pString   /* Pointer to a text string that ends with a null character. */
```

BKM_SETTABTEXT - Remarks

The text is centered from the tab edges.

The application can define a mnemonic key when sending this message by placing a tilde (~) character before the character that is to be the mnemonic key. The notebook brings this page to the top whenever the user presses the mnemonic key.

The mnemonic key processing is not case-sensitive, so the user can type the mnemonic character in either upper or lower case.

The application can remove or change the mnemonic key by sending additional BKM_SETTABTEXT messages for the specified page.

If this message is sent for a page having both major and minor tab attributes, the notebook sets both the major and minor tab text.

BKM_SETTABTEXT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

BKM_SETTABTEXT - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

BKM_TURNTOPAGE

BKM_TURNTOPAGE Field - ulPageId

ulPageId ([ULONG](#))
Page ID.
The page identifier that is to become the top page.

BKM_TURNTOPAGE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

BKM_TURNTOPAGE Return Value - fSuccess

fSuccess ([BOOL](#))
Success indicator.

TRUE	The page was successfully moved to the top of the notebook.
FALSE	Unable to move the page to the top of the notebook. This value is returned if the page ID specified in the <i>ulPageId</i> parameter is invalid.

BKM_TURNTOPAGE - Parameters

ulPageId (ULONG)
Page ID.

The page identifier that is to become the top page.

ulReserved (ULONG)
Reserved value, should be 0.

fSuccess (BOOL)
Success indicator.

TRUE

The page was successfully moved to the top of the notebook.

FALSE

Unable to move the page to the top of the notebook. This value is returned if the page ID specified in the *ulPageId* parameter is invalid.

BKM_TURNTOPAGE - Syntax

This message brings the specified page to the top of the notebook.

```
param1
    ULONG    ulPageId    /* Page ID. */

param2
    ULONG    ulReserved  /* Reserved value, should be 0. */
```

BKM_TURNTOPAGE - Remarks

The application receives a BKN_PAGESELECTED notification code when the new page is brought to the top of the notebook.

BKM_TURNTOPAGE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

BKM_TURNTOPAGE - Topics

Select an item:

WM_CHAR (in Notebook Controls)

WM_CHAR (in Notebook Controls) - Syntax

For the cause of this message, see [WM_CHAR](#).

For a description of the parameters, see [WM_CHAR](#).

WM_CHAR (in Notebook Controls) - Remarks

If the application page window has the focus (for example, the cursor is on a control within the top page dialog), the notebook handles the following keyboard interaction:

Alt+Up Arrow	Sets the focus to the notebook window.
--------------	--

If the notebook control has the focus (for example, the cursor is on the major tab, minor tab or page turning button), the notebook handles the following keyboard interactions:

Alt+Down Arrow	Sets the focus to the application page window.
----------------	--

Tab	Moves the selection cursor to the next position or control.
-----	---

Shift+Tab	Moves the selection cursor to the previous position or control.
-----------	---

Down Arrow or Right Arrow	Moves the selection cursor to the next major or minor tab. If either of these keys is pressed while the selection cursor is on a major tab, the cursor moves to the next major tab. If either of these keys is pressed while the selection cursor is on a minor tab, the cursor moves to the next minor tab. If the next tab is not visible, the tabs are scrolled to bring the next tab into view. If the end of the tabs is reached, scrolling ends.
---------------------------	--

Up Arrow or Left Arrow	Moves the selection cursor to the previous major or minor tab. If either of these keys is pressed while the selection cursor is on a major tab, the cursor moves to the previous major tab. If either of these keys is pressed while the selection cursor is on a minor tab, the cursor moves to the previous minor tab. If the previous tab is not visible, the tabs are scrolled to bring the previous tab into view. If the beginning of the tabs is reached, scrolling ends.
------------------------	--

Enter or Spacebar	The cursored tab page becomes the top page of the notebook.
-------------------	---

Mnemonics	Brings the page whose tab contains the mnemonic character to the top of the notebook whenever the user presses the mnemonic key. Mnemonic key definition is provided by using the BKM_SETTABTEXT message. Coding a mnemonic character (~) before a text character in the BKM_SETTABTEXT message causes that character to be underlined in the tab's text string and activates it as a mnemonic selection character. The mnemonic key pressing is not case-sensitive, so the user can type the mnemonic character in either upper or lower case.
-----------	---

PgDn or Alt+PgDn	Brings the next page to the top of the notebook and sets the selection cursor on the associated tab, if there is one.
PgUp or Alt+PgUp	Brings the previous page to the top of the notebook and sets the selection cursor on the associated tab, if there is one.
Home	Brings the first page of the notebook to the top and sets the selection cursor on the associated tab, if there is one.
End	Brings the last page of the notebook to the top and sets the selection cursor on the associated tab, if there is one.

WM_CHAR (in Notebook Controls) - Default Processing

For a description of the default processing, see [WM_CHAR](#).

WM_CHAR (in Notebook Controls) - Topics

Select an item:
[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_PRESPARAMCHANGED (in Notebook Controls)

WM_PRESPARAMCHANGED (in Notebook Controls) Field - attrtype

attrtype ([ULONG](#))
Attribute type.

Presentation parameter attribute identity.

PP_BACKGROUND_COLOR or PP_BACKGROUND_COLOR_INDEX
Sets the background color of the notebook window. This color is initially set to SYSCLR_FIELD_BACKGROUND.

PP_BORDER_COLOR or PP_BORDER_COLOR_INDEX
Sets the color of the notebook outline. This color is initially set to SYSCLR_WINDOW_FRAME.

PP_FOREGROUND_COLOR or PP_FOREGROUND_COLOR_INDEX
Sets the color of text on the status line. This color is initially set to SYSCLR_WINDOW_TEXT.

PP_HILITEBACKGROUND_COLOR or PP_HILITEBACKGROUND_COLOR_INDEX
Sets the color of the selection cursor. This color is initially set to SYSCLR_HILITEBACKGROUND.

WM_PRESPARAMCHANGED (in Notebook Controls) Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_PRESPARAMCHANGED (in Notebook Controls) Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_PRESPARAMCHANGED (in Notebook Controls) - Parameters

attrtype (ULONG)
Attribute type.

Presentation parameter attribute identity.

PP_BACKGROUND_COLOR or PP_BACKGROUND_COLOR_INDEX
Sets the background color of the notebook window. This color is initially set to SYSCLR_FIELDBACKGROUND.

PP_BORDER_COLOR or PP_BORDER_COLOR_INDEX
Sets the color of the notebook outline. This color is initially set to SYSCLR_WINDOWFRAME.

PP_FOREGROUND_COLOR or PP_FOREGROUND_COLOR_INDEX
Sets the color of text on the status line. This color is initially set to SYSCLR_WINDOWTEXT.

PP_HILITEBACKGROUND_COLOR or PP_HILITEBACKGROUND_COLOR_INDEX
Sets the color of the selection cursor. This color is initially set to SYSCLR_HILITEBACKGROUND.

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

WM_PRESPARAMCHANGED (in Notebook Controls) -

Syntax

For the cause of this message, see [WM_PRESPARAMCHANGED](#).

```
param1
    ULONG attrtype    /* Attribute type. */

param2
    ULONG ulReserved  /* Reserved value, should be 0. */
```

WM_PRESPARAMCHANGED (in Notebook Controls) - Remarks

The application uses this message to notify the notebook that a given inherited presentation parameter has changed.

WM_PRESPARAMCHANGED (in Notebook Controls) - Default Processing

For a description of the default processing, see [WM_PRESPARAMCHANGED](#).

WM_PRESPARAMCHANGED (in Notebook Controls) - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

WM_SIZE (in Notebook Controls)

WM_SIZE (in Notebook Controls) - Syntax

For the cause of this message, see [WM_SIZE](#).

For a description of the parameters, see [WM_SIZE](#).

WM_SIZE (in Notebook Controls) - Remarks

When the size of the notebook window changes, all of the regions are recalculated. The notebook sends a BKN_NEWPAGESIZE notification code to the application. The notebook sets the position and size of application page windows that are associated with pages for whom the BKA_AUTOPAGESIZE attribute is set.

WM_SIZE (in Notebook Controls) - Default Processing

For a description of the default processing, see [WM_SIZE](#).

WM_SIZE (in Notebook Controls) - Topics

- Select an item:
- [Syntax](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

Scroll Bar Control Window Processing

This system-provided window procedure processes the actions on a scroll bar control (WC_SCROLLBAR).

Purpose

Scroll bars are controls used to indicate that additional information can be displayed in a window, logically to the left or right for horizontal scroll bars, logically above or below for vertical scroll bars. The user interface for scroll bars allows for scrolling one unit or one page at a time, or alternatively picking up the scroll bar slider and moving it to a position in the scroll bar that indicates a logical position in the data.

Scroll Bar Control Data

See [SBCDATA](#) for information on scroll bar control data.

Scroll Bar Control Styles

These scroll bar control styles are available:

- | | |
|----------|---------------------------------|
| SBS_HORZ | Create a horizontal scroll bar. |
| SBS_VERT | Create a vertical scroll bar. |

SBS_THUMBSIZE	Indicates the presence of the <i>cVisible</i> and <i>cTotal</i> parameters in the SBCDATA data structure.
SBS_AUTOTRACK	The slider scrolls as more information is being displayed on the screen.
SBS_AUTOSIZE	The scroll bar slider changes size to reflect the amount of data contained in the window.

Default Colors

The following system colors are used when the system draws scroll-bar controls:

```
SYSCLR_SCROLLBAR
SYSCLR_WINDOWFRAME
SYSCLR_FIELDBACKGROUND
SYSCLR_WINDOW
SYSCLR_BUTTONMIDDLE.
```

Some of these defaults can be replaced by using the following presentation parameters in the application resource script file or source code:

```
PP_FOREGROUNDColor
PP_BORDERColor
PP_HILITEForegroundColor.
```

Scroll Bar System Values

Applications can use the following system values to create and add control scroll bars:

SV_CXVSCROLL	Width of the vertical scroll-bar.
SV_CYHSCROLL	Height of the horizontal scroll-bar.
SV_CYVSCROLLARROW	Height of the vertical scroll-bar arrow bit maps.
SV_CXHSCROLLARROW	Height of the horizontal scroll-bar arrow bit maps.
SV_FIRSTSCROLLRATE	The delay (in milliseconds) before autoscrolling starts, when using a scroll bar.
SV_SCROLLRATE	The delay (in milliseconds) between scroll operations, when using a scroll bar.
SYSCLR_SCROLLBAR	Color for drawing scroll-bar backgrounds.
TID_SCROLL	Timer ID for a reserved scrolling time. This is used for sending notification messages when a scroll-arrow or scroll-bar background is selected.

Scroll Bar Control Notification Messages

These messages are initiated by the scroll bar control window procedure to notify its owner of significant events.

WM_HSCROLL (in Horizontal Scroll Bars)

WM_HSCROLL (in Horizontal Scroll Bars) - Syntax

For the cause of this message, see [WM_HSCROLL](#).

For a description of the parameters, see [WM_HSCROLL](#).

WM_HSCROLL (in Horizontal Scroll Bars) - Remarks

The scroll bar control window procedure generates this message and posts it to its owner, informing the owner of the event.

WM_HSCROLL (in Horizontal Scroll Bars) - Default Processing

The default window procedure takes no action on this message, other than to set *uiReserved* to 0.

WM_HSCROLL (in Horizontal Scroll Bars) - Related Messages

Related Messages

- [WM_HSCROLL](#)
-

WM_HSCROLL (in Horizontal Scroll Bars) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_VSCROLL (in Vertical Scroll Bars)

WM_VSCROLL (in Vertical Scroll Bars) - Syntax

For the cause of this message, see [WM_VSCROLL](#).

For a description of the parameters, see [WM_VSCROLL](#).

WM_VSCROLL (in Vertical Scroll Bars) - Remarks

The scroll bar control window procedure generates this message and posts the message to the owner of the procedure, informing the owner of the event.

WM_VSCROLL (in Vertical Scroll Bars) - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_VSCROLL (in Vertical Scroll Bars) - Related Messages

Related Messages

- [WM_VSCROLL](#)
-

WM_VSCROLL (in Vertical Scroll Bars) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

Scroll Bar Control Window Messages

This section describes the scroll bar control window procedure actions on receiving the following messages.

SBM_QUERYPOS

SBM_QUERYPOS Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

SBM_QUERYPOS Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

SBM_QUERYPOS Return Value - sslider

sslider (SHORT)
Slider position.

SBM_QUERYPOS - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

sslider (SHORT)
Slider position.

SBM_QUERYPOS - Syntax

This message returns the current slider position in a scroll bar window.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */
```

```
param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

SBM_QUERYPOS - Remarks

The scroll bar control window procedure responds to this message by returning the current slider position.

SBM_QUERYPOS - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *sslider* to the default value of 0.

SBM_QUERYPOS - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

SBM_QUERYRANGE

SBM_QUERYRANGE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

SBM_QUERYRANGE Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

SBM_QUERYRANGE Field - sfirst

sfirst ([SHORT](#))
First bound.

SBM_QUERYRANGE Field - slast

slast ([SHORT](#))
Last bound.

SBM_QUERYRANGE - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

sfirst ([SHORT](#))
First bound.

slast ([SHORT](#))
Last bound.

SBM_QUERYRANGE - Syntax

This message returns the scroll bar range minimum and maximum values.

```
param1
    ULONG    ulReserved /* Reserved value, should be 0. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */

returns
    SHORT    sfirst     /* First bound. */
    SHORT    slast      /* Last bound. */
```

SBM_QUERYRANGE - Remarks

The scroll bar control window procedure responds to this message by returning the first and last bounds of the scroll bar range.

SBM_QUERYRANGE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *ReturnCode* to the default value of *sfirst* and *slast* to 0.

SBM_QUERYRANGE - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

SBM_SETPOS

SBM_SETPOS Field - sslider

sslider ([SHORT](#))

Position of slider.

If this value is outside the scroll-bar range, the slider is moved to the nearest valid position within the range.

SBM_SETPOS Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

SBM_SETPOS Return Value - rc

rc (**BOOL**)

Success indicator.

TRUE

Successful completion

FALSE

Error occurred

SBM_SETPOS - Parameters

slider (**SHORT**)

Position of slider.

If this value is outside the scroll-bar range, the slider is moved to the nearest valid position within the range.

ulReserved (**ULONG**)

Reserved value, should be 0.

rc (**BOOL**)

Success indicator.

TRUE

Successful completion

FALSE

Error occurred

SBM_SETPOS - Syntax

This message sets the position of the slider in a scroll bar window.

```
param1
    SHORT  slider    /* Position of slider. */

param2
    ULONG  ulReserved /* Reserved value, should be 0. */
```

SBM_SETPOS - Remarks

The scroll bar control window procedure responds to this message by setting the position of the slider.

The scroll bar control is redrawn to reflect the change.

SBM_SETPOS - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it.

SBM_SETPOS - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

SBM_SETSCROLLBAR

SBM_SETSCROLLBAR Field - sslider

sslider ([SHORT](#))

Position of slider.

If this value is outside the scroll-bar range, the slider is moved to the nearest valid position within the range.

SBM_SETSCROLLBAR Field - sfirst

sfirst ([SHORT](#))

First bound.

This value must not be less than 0. If a value less than 0 is supplied, 0 is used as the value.

SBM_SETSCROLLBAR Field - slast

slast ([SHORT](#))

Last bound.

The value must not be less than 0 or *sfirst*. If a value less than this is supplied, the higher of 0 or *sfirst* is used as the value.

SBM_SETSCROLLBAR Return Value - rc

rc (BOOL)

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

SBM_SETSCROLLBAR - Parameters

slider (SHORT)

Position of slider.

If this value is outside the scroll-bar range, the slider is moved to the nearest valid position within the range.

sfirst (SHORT)

First bound.

This value must not be less than 0. If a value less than 0 is supplied, 0 is used as the value.

slast (SHORT)

Last bound.

The value must not be less than 0 or *sfirst*. If a value less than this is supplied, the higher of 0 or *sfirst* is used as the value.

rc (BOOL)

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

SBM_SETSCROLLBAR - Syntax

This message sets the scroll-bar range and slider position.

```
param1
    SHORT  slider /* Position of slider. */

param2
    SHORT  sfirst /* First bound. */
    SHORT  slast  /* Last bound. */
```

SBM_SETSCROLLBAR - Remarks

The scroll bar control window procedure responds to this message by setting the values of the information range and the position of the slider.

The scroll bar is redrawn to reflect the change.

For example, if a scroll-bar is to allow scrolling through 100 lines of text, of which 50 are visible at any one time, and the top display line is currently number 25, *sfirst* should be set to 1, *slast* to 51 (since there are only 51 positions at which the slider may be placed), and *sslider* to 25. The [SBM_SETTHUMBSize](#) message should be used in this example to set the slider size to 50 visible parts out of 100.

SBM_SETSCROLLBAR - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it.

SBM_SETSCROLLBAR - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

SBM_SETTHUMBSize

SBM_SETTHUMBSize Field - svisible

svisible ([SHORT](#))

Size of the visible part of the document.

SBM_SETTHUMBSize Field - stotal

stotal ([SHORT](#))
Size of the entire document.

SBM_SETTHUMBSIZE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

SBM_SETTHUMBSIZE Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SBM_SETTHUMBSIZE - Parameters

svisible ([SHORT](#))
Size of the visible part of the document.

stotal ([SHORT](#))
Size of the entire document.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SBM_SETTHUMBSIZE - Syntax

This message sets the scroll bar slider size.

```
param1
    SHORT  svisible    /* Size of the visible part of the document. */
    SHORT  stotal      /* Size of the entire document. */

param2
    ULONG  ulReserved  /* Reserved value, should be 0. */
```

SBM_SETTHUMBSIZE - Remarks

The scroll bar control window procedure responds to this message by setting the size of the slider proportional to the visible part of the document. If the visible part exceeds or is equal to the entire document the scroll bar is disabled, otherwise the scroll bar is enabled.

The scroll bar is redrawn to reflect the change.

SBM_SETTHUMBSIZE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it.

SBM_SETTHUMBSIZE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_QUERYCONVERTPOS (in Scroll Bars)

WM_QUERYCONVERTPOS (in Scroll Bars) - Syntax

For the cause of this message, see [WM_QUERYCONVERTPOS](#).

For a description of the parameters, see [WM_QUERYCONVERTPOS](#).

WM_QUERYCONVERTPOS (in Scroll Bars) - Remarks

The scroll bar control window procedure returns QCP_NOCONVERT.

WM_QUERYCONVERTPOS (in Scroll Bars) - Default Processing

For the default window procedure processing of this message see [WM_QUERYCONVERTPOS](#).

WM_QUERYCONVERTPOS (in Scroll Bars) - Related Messages

Related Messages

- [WM_QUERYCONVERTPOS](#)
-

WM_QUERYCONVERTPOS (in Scroll Bars) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_QUERYWINDOWPARAMS (in Scroll Bars)

WM_QUERYWINDOWPARAMS (in Scroll Bars) - Syntax

This message occurs when an application queries the scroll bar control window parameters.

For a description of the parameters, see [WM_QUERYWINDOWPARAMS](#).

WM_QUERYWINDOWPARAMS (in Scroll Bars) - Remarks

The scroll bar control window procedure responds to this message by returning the window parameters indicated by the *fsStatus* parameter

of the [WNDPARAMS](#) data structure identified by the *pwndparams* parameter.

WM_QUERYWINDOWPARAMS (in Scroll Bars) - Default Processing

The default window procedure sets the *cchText*, *cbPresParams*, and *cbCtlData* parameters of the [WNDPARAMS](#) data structure, identified by *pwndparams*, to 0 and sets *rc* to FALSE.

WM_QUERYWINDOWPARAMS (in Scroll Bars) - Related Messages

Related Messages

- [WM_QUERYWINDOWPARAMS](#)
-

WM_QUERYWINDOWPARAMS (in Scroll Bars) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_SETWINDOWPARAMS (in Scroll Bars)

WM_SETWINDOWPARAMS (in Scroll Bars) - Syntax

This message occurs when an application sets or changes the scroll bar control window parameters.

For a description of the parameters, see [WM_SETWINDOWPARAMS](#).

WM_SETWINDOWPARAMS (in Scroll Bars) - Remarks

The scroll bar control window procedure responds to this message by setting the window parameters indicated by the *fsStatus* parameter of

the [WNDPARAMS](#) data structure identified by the *hwndparams* parameter.

WM_SETWINDOWPARAMS (in Scroll Bars) - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_SETWINDOWPARAMS (in Scroll Bars) - Related Messages

Related Messages

- [WM_SETWINDOWPARAMS](#)
-

WM_SETWINDOWPARAMS (in Scroll Bars) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

Slider Control Window Processing

This system-provided window procedure processes the actions on a slider control (WC_SLIDER).

Purpose

A slider control (WC_SLIDER window class) is a visual component whose specific purpose is to allow a user to set, display, or modify a value by moving a slider arm along a slider shaft. Sliders are typically used to allow a user to easily set values that have familiar increments, such as feet, inches, degrees, decibels, and so forth.

However, they can also be used for other purposes when immediate feedback is necessary, such as to blend colors or to show the percentage of a task that has completed. For example, an application might allow a user to mix and match color shades by moving a slider arm, or a read-only slider could be provided that shows how much of a task has completed by filling in the slider shaft as the task progresses. These are just a few examples to show you the many ways in which sliders can be used.

The appearance of and user interaction for a slider is similar to the appearance of and user interaction for a scroll bar. However, these two controls are not interchangeable because each has a distinct purpose. The scroll bar is used to scroll into view information that is outside a window's client area, while the slider is used to set, display, or modify that information, whether it is in the client area or not in the client area.

The slider is designed to be customizable to meet varying application requirements, while providing an easy-to-use user interface component that can be used to develop products that conform to the Common User Access (CUA) user interface guidelines. The application can specify different scales, sizes, and orientations for its sliders, but the underlying function of the control remains the same. For a complete description of CUA sliders, refer to the *SAA CUA Guide to User Interface Design* and the *SAA CUA Advanced Interface Design Reference*.

Slider Control Styles

Slider control window styles are set when a slider window is created. The following styles can be set when creating a slider control window. If no styles are specified, defaults, which are identified in the following descriptions, are used.

- Specify either of the following to determine the slider's orientation:

SLS_HORIZONTAL

The slider is positioned horizontally. The slider arm can move left and right on the slider shaft. A scale can be placed on top of the slider shaft, below the slider shaft, or in both places. This is the default orientation of the slider.

SLS_VERTICAL

The slider is positioned vertically. The slider arm can move up and down the slider shaft. A scale can be placed on the left side of the slider shaft, on the right side of the slider shaft, or in both places.

- Specify one of the following to position the slider within the slider window:

SLS_CENTER

The slider is centered in the slider window. This is the default positioning of the slider.

SLS_BOTTOM

The slider is positioned at the bottom of the slider window. This is valid for horizontal sliders only.

SLS_TOP

The slider is positioned at the top of the slider window. This is valid for horizontal sliders only.

SLS_LEFT

The slider is positioned at the left edge of the slider window. This is valid for vertical sliders only.

SLS_RIGHT

The slider is positioned at the right edge of the slider window. This is valid for vertical sliders only.

- Specify one of the following to determine the location of the scale on the slider shaft:

SLS_PRIMARYSCALE1

The slider uses the increment and spacing specified for scale 1 as the incremental value for positioning the slider arm. Scale 1 is displayed above the slider shaft of a horizontal slider and to the right of the slider shaft of a vertical slider. This is the default for a slider.

SLS_PRIMARYSCALE2

The slider uses the increment and spacing specified for scale 2 as the incremental value for positioning the slider arm. Scale 2 is displayed below the slider shaft of a horizontal slider and to the left of the slider shaft of a vertical slider.

- Specify one of the following to determine the slider arm's home position:

SLS_HOMELEFT

The slider uses the left edge of the slider as the base value for incrementing. This is the default for horizontal sliders and is valid for horizontal sliders only.

SLS_HOMERIGHT

The slider uses the right edge of the slider as the base value for incrementing. This is valid for horizontal sliders only.

SLS_HOMEBOTTOM

The slider uses the bottom of the slider as the base value for incrementing. This is the default for vertical sliders and is valid for vertical sliders only.

SLS_HOMETOP

The slider uses the top of the slider as the base value for incrementing. This is valid for vertical sliders only.

- Specify one of the following to determine the location of the slider buttons. If you do not specify one of these styles, or if conflicting styles are specified, slider buttons are not included in the slider control.

SLS_BUTTONSLEFT

The slider includes incremental slider buttons with the control and places them to the left of the slider shaft. These slider buttons move the slider arm by one position, either left or right, in the direction that is selected. This is valid for horizontal sliders only.

SLS_BUTTONSRIGHT

The slider includes incremental slider buttons with the control and places them to the right of the slider shaft. These slider buttons move the slider arm by one position, either left or right, in the direction that is selected. This is valid for horizontal sliders only.

SLS_BUTTONSBOTTOM

The slider includes incremental slider buttons with the control and places them at the bottom of the slider shaft. These slider buttons move the slider arm by one position, either up or down, in the direction that is selected. This is valid for vertical sliders only.

SLS_BUTTONSTOP

The slider includes incremental slider buttons with the control and places them at the top of the slider shaft. These slider buttons move the slider arm by one position, either up or down, in the direction that is selected. This is valid for vertical sliders only.

- Other styles that you can specify:

SLS_SNAPTOINCREMENT

The slider arm, when moved to a position between two specified values on the slider scale, such as between two tick marks, is positioned on the nearest value and is redrawn at that position. If this style is not specified, the slider arm remains at the position to which it is moved.

SLS_READONLY

The slider is created as a read-only slider. This means that the user cannot interact with the slider. It is used merely as a mechanism to present a quantity to the user, such as the percentage of completion of an ongoing task. Visual differences for a read-only slider include a narrow slider arm, no slider buttons and no detents.

SLS_RIBBONSTRIP

As the slider arm moves, the slider fills the slider shaft between the home position and the slider arm with a color value that is different from the slider shaft color, similar to the mercury in a thermometer.

SLS_OWNERDRAW

The application is notified whenever the slider shaft, the ribbon strip, the slider arm, and the slider background are to be drawn.

Slider Control Data

See [SLDCDATA](#).

Slider Control Notification Messages

These messages are initiated by the slider control window to notify its owner of significant events.

WM_CONTROL (in Slider Controls)

WM_CONTROL (in Slider Controls) Field - id

id ([USHORT](#))
Slider control identity.

WM_CONTROL (in Slider Controls) Field - notifycode

notifycode ([USHORT](#))
Notification code.

The slider control uses these notification codes:

SLN_CHANGE	The slider arm position has changed.
SLN_KILLFOCUS	The slider control is losing the focus.
SLN_SETFOCUS	The slider control is receiving the focus.
SLN_SLIDERTRACK	The slider arm is being dragged, but has not been released.

WM_CONTROL (in Slider Controls) Field - notifyinfo

notifyinfo ([ULONG](#))
Control-specific information.

When the value of the *notifycode* parameter is SLN_CHANGE or SLN_SLIDERTRACK, this value is the new arm position, expressed as the number of pixels from the home position.

Otherwise, this value is the window handle ([HWND](#)) of the slider control.

WM_CONTROL (in Slider Controls) Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_CONTROL (in Slider Controls) - Parameters

id ([USHORT](#))
Slider control identity.

notifycode ([USHORT](#))

Notification code.

The slider control uses these notification codes:

SLN_CHANGE

The slider arm position has changed.

SLN_KILLFOCUS

The slider control is losing the focus.

SLN_SETFOCUS

The slider control is receiving the focus.

SLN_SLIDERTRACK

The slider arm is being dragged, but has not been released.

notifyinfo ([ULONG](#))

Control-specific information.

When the value of the *notifycode* parameter is **SLN_CHANGE** or **SLN_SLIDERTRACK**, this value is the new arm position, expressed as the number of pixels from the home position.

Otherwise, this value is the window handle ([HWND](#)) of the slider control.

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_CONTROL (in Slider Controls) - Syntax

For the cause of this message, see [WM_CONTROL](#).

```
param1
    USHORT id           /* Slider control identity. */
    USHORT notifycode   /* Notification code. */

param2
    ULONG notifyinfo    /* Control-specific information. */
```

WM_CONTROL (in Slider Controls) - Remarks

The slider control window procedure generates this message and sends it to its owner, informing the owner of this event.

WM_CONTROL (in Slider Controls) - Default Processing

For a description of the default processing, see [WM_CONTROL](#).

WM_CONTROL (in Slider Controls) - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

WM_CONTROLPOINTER (in Slider Controls)

WM_CONTROLPOINTER (in Slider Controls) - Syntax

For the cause of this message, see [WM_CONTROLPOINTER](#).

For a description of the parameters, see [WM_CONTROLPOINTER](#).

WM_CONTROLPOINTER (in Slider Controls) - Remarks

For the appropriate remarks, see [WM_CONTROLPOINTER](#).

WM_CONTROLPOINTER (in Slider Controls) - Default Processing

For the default processing, see [WM_CONTROLPOINTER](#).

WM_CONTROLPOINTER (in Slider Controls) - Topics

Select an item:

[Syntax](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

WM_DRAWITEM (in Slider Controls)

WM_DRAWITEM (in Slider Controls) Field - id

id ([USHORT](#))
Window identifier.

The window identifier of the slider control sending this notification message.

WM_DRAWITEM (in Slider Controls) Field - powneritem

powneritem ([POWNERITEM](#))
Pointer to an [OWNERITEM](#) data structure.

The following list defines the [OWNERITEM](#) data structure fields that apply to the slider control. See [OWNERITEM](#) for the default field values.

hwnd ([HWND](#))
Slider window handle.

hps ([HPS](#))
Presentation-space handle.

fsState ([ULONG](#))
Slider window style flags. See [Slider Control Styles](#) for descriptions of these style flags.

fsAttribute ([ULONG](#))
Reserved.

fsStateOld ([ULONG](#))
Reserved.

fsAttributeOld ([ULONG](#))
Reserved.

rcItem ([RECTL](#))
Item rectangle to be drawn in window coordinates.

idItem ([LONG](#))
Identity of item to be drawn:

SDA_SLIDERSHAFT	Specifies that the slider shaft is to be drawn.
SDA_RIBBONSTRIP	Specifies that the slider shaft area that contains a ribbon strip is to be drawn.
SDA_SLIDERARM	Specifies that the slider arm is to be drawn.
SDA_BACKGROUND	Specifies that the slider background is to be drawn.

hItem ([ULONG](#))
Reserved.

WM_DRAWITEM (in Slider Controls) Return Value - rc

rc (BOOL)
Item-drawn indicator.

TRUE The owner draws the item.

FALSE If the owner does not draw the item, the owner returns this value and the slider control draws the item.

WM_DRAWITEM (in Slider Controls) - Parameters

id (USHORT)
Window identifier.

The window identifier of the slider control sending this notification message.

powneritem (POWNERITEM)
Pointer to an OWNERITEM data structure.

The following list defines the OWNERITEM data structure fields that apply to the slider control. See OWNERITEM for the default field values.

- hwnd* (HWND) Slider window handle.
- hps* (HPS) Presentation-space handle.
- fsState* (ULONG) Slider window style flags. See Slider Control Styles for descriptions of these style flags.
- fsAttribute* (ULONG) Reserved.
- fsStateOld* (ULONG) Reserved.
- fsAttributeOld* (ULONG) Reserved.
- rcItem* (RECT) Item rectangle to be drawn in window coordinates.
- idItem* (LONG)
- | | |
|-----------------|---|
| SDA_SLIDERSHAFT | Specifies that the slider shaft is to be drawn. |
| SDA_RIBBONSTRIP | Specifies that the slider shaft area that contains a ribbon strip is to be drawn. |
| SDA_SLIDERARM | Specifies that the slider arm is to be drawn. |
| SDA_BACKGROUND | Specifies that the slider background is to be drawn. |
- hlItem* (ULONG) Reserved.

rc (BOOL)
Item-drawn indicator.

TRUE The owner draws the item.

FALSE If the owner does not draw the item, the owner returns this value and the slider control draws the item.

WM_DRAWITEM (in Slider Controls) - Syntax

If the SLS_OWNERDRAW style bit is set for a slider control, this notification message is sent to that slider control's owner whenever the slider shaft, ribbon strip, slider arm, and slider background are to be drawn.

```
param1
    USHORT      id          /* Window identifier. */

param2
    POWNERITEM  powneritem /* Pointer to an OWNERITEM data structure. */
```

WM_DRAWITEM (in Slider Controls) - Remarks

The slider control provides this message to give the application the opportunity to provide a custom slider shaft, custom ribbon strip, custom slider arm, and custom background. The application can specify one or all of these items and is given the opportunity to do so.

The slider control window procedure generates this message and sends it to its owner, informing the owner that an item is to be drawn. The owner is then given the opportunity to draw that item, and to indicate that an item has been drawn or that the slider control is to draw it.

WM_DRAWITEM (in Slider Controls) - Default Processing

For a description of the default processing, see [WM_DRAWITEM](#).

WM_DRAWITEM (in Slider Controls) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

Slider Control Window Messages

This section describes the slider control window procedure actions on receiving the following messages.

SLM_ADDDETENT

SLM_ADDDETENT Field - usDetentPos

usDetentPos ([USHORT](#))

Detent position.

Number of pixels the detent is positioned from home.

SLM_ADDDETENT Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

SLM_ADDDETENT Return Value - ulDetentId

ulDetentId ([ULONG](#))

Detent ID.

Unique identifier for the detent being added to the slider. If 0 is returned, an error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_HEAP_MAX_SIZE_REACHED
- PMERR_PARAMETER_OUT_OF_RANGE.

SLM_ADDDETENT - Parameters

usDetentPos ([USHORT](#))

Detent position.

Number of pixels the detent is positioned from home.

ulReserved ([ULONG](#))

Reserved value, should be 0.

ulDetentId ([ULONG](#))

Detent ID.

Unique identifier for the detent being added to the slider. If 0 is returned, an error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_HEAP_MAX_SIZE_REACHED
- PMERR_PARAMETER_OUT_OF_RANGE.

SLM_ADDDETENT - Syntax

This message places a detent along the slider shaft at the position specified on the primary scale. A detent is an indicator that represents a predefined value for a quantity. It does not have to correspond to an increment of the slider.

```
param1
    USHORT  usDetentPos  /*  Detent position. */

param2
    ULONG   ulReserved  /*  Reserved value, should be 0. */
```

SLM_ADDDETENT - Remarks

The application uses this message to add detents along the slider to denote values that do not fall along an increment setting. An example of this would be a slider that represents temperature and has increments that are on multiples of 5. A detent could be located at 32, instead of 30 or 35, for special purposes.

SLM_ADDDETENT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

SLM_ADDDETENT - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

SLM_QUERYDETENTPOS

SLM_QUERYDETENTPOS Field - ulDetentId

ulDetentId (ULONG)
Detent ID.

Unique detent identifier, which indicates the position to be returned.

SLM_QUERYDETENTPOS Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

SLM_QUERYDETENTPOS Field - usDetentPos

usDetentPos (USHORT)
Detent position.

Number of pixels the detent is positioned from home.

>= 0
Number of pixels the detent is positioned from home.

SLDERR_INVALID_PARAMETERS
An error occurred. The [WinGetLastError](#) function may return the following error:

PMERR_INVALID_PARAMETERS.

SLM_QUERYDETENTPOS Field - fDetentLocation

fDetentLocation (USHORT)
Scale.

The scale along which the detent is located. One of the following:

SMA_SCALE1
Detent position is along scale 1.

SMA_SCALE2
Detent position is along scale 2.

SLM_QUERYDETENTPOS - Parameters

ulDetentId (ULONG)
Detent ID.

Unique detent identifier, which indicates the position to be returned.

ulReserved (ULONG)

Reserved value, should be 0.

usDetentPos (USHORT)

Detent position.

Number of pixels the detent is positioned from home.

≥ 0

Number of pixels the detent is positioned from home.

SLDERR_INVALID_PARAMETERS

An error occurred. The [WinGetLastError](#) function may return the following error:

PMERR_INVALID_PARAMETERS.

fDetentLocation (USHORT)

Scale.

The scale along which the detent is located. One of the following:

SMA_SCALE1

Detent position is along scale 1.

SMA_SCALE2

Detent position is along scale 2.

SLM_QUERYDETENTPOS - Syntax

This message queries for the current position of a detent.

```
param1
    ULONG    ulDetentId        /* Detent ID. */

param2
    ULONG    ulReserved        /* Reserved value, should be 0. */

returns
    USHORT    usDetentPos      /* Detent position. */
    USHORT    fDetentLocation  /* Scale. */
```

SLM_QUERYDETENTPOS - Remarks

An application could use this message to place text above the detent or position an item relative to it.

SLM_QUERYDETENTPOS - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

SLM_QUERYDETENTPOS - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

SLM_QUERYSCALETEXT

SLM_QUERYSCALETEXT Field - usTickNum

usTickNum ([USHORT](#))
Tick location.

Tick location to query for the text.

SLM_QUERYSCALETEXT Field - usBufLen

usBufLen ([USHORT](#))
Buffer length.

Length of the buffer to copy the text into. The buffer size should include space for the null termination character.

SLM_QUERYSCALETEXT Field - pTickText

pTickText ([PSZ](#))
Pointer to the buffer into which to place the text string for the tick mark.

SLM_QUERYSCALETEXT Return Value - sTextLen

sTextLen ([SHORT](#))

Count of bytes.

Count of bytes copied to buffer.

>= 0

Length of the text string, excluding the null termination character.

SLDERR_INVALID_PARAMETERS

An error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_PARAMETER_OUT_OF_RANGE.

SLM_QUERYSCALETEXT - Parameters

usTickNum ([USHORT](#))

Tick location.

Tick location to query for the text.

usBufLen ([USHORT](#))

Buffer length.

Length of the buffer to copy the text into. The buffer size should include space for the null termination character.

pTickText ([PSZ](#))

Pointer to the buffer into which to place the text string for the tick mark.

sTextLen ([SHORT](#))

Count of bytes.

Count of bytes copied to buffer.

>= 0

Length of the text string, excluding the null termination character.

SLDERR_INVALID_PARAMETERS

An error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_PARAMETER_OUT_OF_RANGE.

SLM_QUERYSCALETEXT - Syntax

This message queries for the text associated with a tick mark for the primary scale and copies that text into a buffer.

```
param1
    USHORT  usTickNum    /* Tick location. */
    USHORT  usBufLen     /* Buffer length. */

param2
    PSZ      pTickText   /* Pointer to the buffer into which to place the text string for the tick mark. */
```

SLM_QUERYSCALETEXT - Remarks

This message could be used to return text that represents the current position of the slider arm or to query the text for use in ownerdraw mode.

By specifying 0 as the value of the *usBufLen* parameter and then looking at the value returned in the *sTextLen* parameter, an application can determine the size of the buffer to allocate for copying the text. An application can then allocate a buffer of this size, adding one byte for the null termination character, and then specify this buffer and size on the query call.

SLM_QUERYSCALETEXT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

SLM_QUERYSCALETEXT - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

SLM_QUERYSLIDERINFO

SLM_QUERYSLIDERINFO Field - usInfoType

usInfoType ([USHORT](#))

Information attribute.

Attribute that identifies the requested information. It can be one of the following:

SMA_SHAFTDIMENSIONS

Queries for the length and breadth of the slider shaft.

SMA_SHAFTPOSITION

Queries for the x-, y-position of the lower-left corner of the slider shaft.

SMA_SLIDERARMDIMENSIONS

Queries for the length and breadth of the slider arm.

SMA_SLIDERARMPOSITION

Queries for the position of the slider arm. The position can be returned either as an increment position or a range

value.

SLM_QUERYSLIDERINFO Field - usArmPosType

usArmPosType (USHORT)

Format attribute.

Attribute that identifies the format in which the information should be returned if the slider arm position is requested. This value is ignored for all other queries and is one of the following:

SMA_RANGEVALUE

The value returned in the low order word represents the number of pixels between the home position and the current arm position. The high order word represents the pixel count of the entire range of the slider control.

SMA_INCREMENTVALUE

The value returned represents an increment position using the primary scale.

SLM_QUERYSLIDERINFO Field - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

SLM_QUERYSLIDERINFO Return Value - ulInfo

ulInfo (ULONG)

Return information.

One of the following items, depending on which SMA_* message attribute or attributes, were set with the [SLM_SETSLIDERINFO](#) message:

- If the SMA_SHAFTDIMENSIONS attribute is set, the following is returned:

usShaftLength (USHORT)

Length of the slider shaft, in pixels. It is the width of the slider shaft for horizontal sliders, and the height of the slider shaft for vertical sliders.

usShaftBreadth (USHORT)

Breadth of the slider shaft, in pixels. It is the height of the slider shaft for horizontal sliders, and the width of the slider shaft for vertical sliders.

- If the SMA_SHAFTPOSITION attribute is set, the following is returned:

xShaftCoord (USHORT)

X-coordinate of the slider shaft position within the slider window. This value is expressed in window coordinates and represents the lower-left corner of the slider shaft.

yShaftCoord (USHORT)

Y-coordinate of the slider shaft position within the slider window. This value is expressed in window coordinates and represents the lower-left corner of the slider shaft.

- If the SMA_SLIDERARMDIMENSIONS attribute is set, the following is returned:

usArmLength ([USHORT](#))

Length of the slider arm, in pixels. It is the width of the slider arm for horizontal sliders and the height of the slider arm for vertical sliders.

usArmBreadth ([USHORT](#))

Breadth of the slider arm, in pixels. It is the height of the slider arm for horizontal sliders and the width of the slider arm for vertical sliders.

- If the SMA_SLIDERARMPOSITION and SMA_RANGEVALUE attributes are set, the following is returned:

usArmPos ([USHORT](#))

Number of pixels from the home position to the slider arm.

usSliderRange ([USHORT](#))

Number of pixels over which the user could select a value on the slider.

- If the SMA_SLIDERARMPOSITION and SMA_INCREMENTVALUE attributes are set, the following is returned:

usIncrementPos ([USHORT](#))

Increment that corresponds to the current position of the slider arm.

- If the SLDERR_INVALID_PARAMETERS error is returned, an error occurred. The [WinGetLastError](#) function may return the following error:

PMERR_INVALID_PARAMETERS.

SLM_QUERYSLIDERINFO - Parameters

usInfoType ([USHORT](#))

Information attribute.

Attribute that identifies the requested information. It can be one of the following:

SMA_SHAFTDIMENSIONS

Queries for the length and breadth of the slider shaft.

SMA_SHAFTPOSITION

Queries for the x-, y-position of the lower-left corner of the slider shaft.

SMA_SLIDERARMDIMENSIONS

Queries for the length and breadth of the slider arm.

SMA_SLIDERARMPOSITION

Queries for the position of the slider arm. The position can be returned either as an increment position or a range value.

usArmPosType ([USHORT](#))

Format attribute.

Attribute that identifies the format in which the information should be returned if the slider arm position is requested. This value is ignored for all other queries and is one of the following:

SMA_RANGEVALUE

The value returned in the low order word represents the number of pixels between the home position and the current arm position. The high order word represents the pixel count of the entire range of the slider control.

SMA_INCREMENTVALUE

The value returned represents an increment position using the primary scale.

ulReserved ([ULONG](#))

Reserved value, should be 0.

ulInfo ([ULONG](#))

Return information.

One of the following items, depending on which SMA_* message attribute or attributes, were set with the [SLM_SETSLIDERINFO](#) message:

- If the SMA_SHAFTDIMENSIONS attribute is set, the following is returned:
 - usShaftLength* ([USHORT](#))
Length of the slider shaft, in pixels. It is the width of the slider shaft for horizontal sliders, and the height of the slider shaft for vertical sliders.
 - usShaftBreadth* ([USHORT](#))
Breadth of the slider shaft, in pixels. It is the height of the slider shaft for horizontal sliders, and the width of the slider shaft for vertical sliders.
- If the SMA_SHAFTPOSITION attribute is set, the following is returned:
 - xShaftCoord* ([USHORT](#))
X-coordinate of the slider shaft position within the slider window. This value is expressed in window coordinates and represents the lower-left corner of the slider shaft.
 - yShaftCoord* ([USHORT](#))
Y-coordinate of the slider shaft position within the slider window. This value is expressed in window coordinates and represents the lower-left corner of the slider shaft.
- If the SMA_SLIDERARMDIMENSIONS attribute is set, the following is returned:
 - usArmLength* ([USHORT](#))
Length of the slider arm, in pixels. It is the width of the slider arm for horizontal sliders and the height of the slider arm for vertical sliders.
 - usArmBreadth* ([USHORT](#))
Breadth of the slider arm, in pixels. It is the height of the slider arm for horizontal sliders and the width of the slider arm for vertical sliders.
- If the SMA_SLIDERARMPOSITION and SMA_RANGEVALUE attributes are set, the following is returned:
 - usArmPos* ([USHORT](#))
Number of pixels from the home position to the slider arm.
 - usSliderRange* ([USHORT](#))
Number of pixels over which the user could select a value on the slider.
- If the SMA_SLIDERARMPOSITION and SMA_INCREMENTVALUE attributes are set, the following is returned:
 - usIncrementPos* ([USHORT](#))
Increment that corresponds to the current position of the slider arm.
- If the SLDERR_INVALID_PARAMETERS error is returned, an error occurred. The [WinGetLastError](#) function may return the following error:

PMERR_INVALID_PARAMETERS.

SLM_QUERYSLIDERINFO - Syntax

This message queries the current position or dimensions of a key component of the slider. The information returned and its format depends on the type of information requested.

```
param1
    USHORT  usInfoType    /* Information attribute. */
    USHORT  usArmPosType  /* Format attribute. */

param2
    ULONG   ulReserved    /* Reserved value, should be 0. */
```

SLM_QUERYSLIDERINFO - Remarks

The application uses this message to query for information about individual parts of a slider control, or the value selected by a user.

SLM_QUERYSLIDERINFO - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

SLM_QUERYSLIDERINFO - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

SLM_QUERYTICKPOS

SLM_QUERYTICKPOS Field - usTickNum

usTickNum ([USHORT](#))

Tick mark location.

Specifies the tick mark location to query for the position.

SLM_QUERYTICKPOS Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

SLM_QUERYTICKPOS Field - xTickPos

xTickPos (USHORT)
X-coordinate.

X-coordinate of the point that represents the position of the tick mark. It is the starting position of the tick mark and represents the end of the tick mark closest to the slider shaft.

SLM_QUERYTICKPOS Field - yTickPos

yTickPos (USHORT)
Y-coordinate.

Y-coordinate of the point that represents the position of the tick mark. It is the starting position of the tick mark and represents the end of the tick mark closest to the slider shaft.

If NULL is returned in either parameter, an error occurred. The [WinGetLastError](#) function may return the following error:

PMERR_PARAMETER_OUT_OF_RANGE.

SLM_QUERYTICKPOS - Parameters

usTickNum (USHORT)
Tick mark location.

Specifies the tick mark location to query for the position.

ulReserved (ULONG)
Reserved value, should be 0.

xTickPos (USHORT)
X-coordinate.

X-coordinate of the point that represents the position of the tick mark. It is the starting position of the tick mark and represents the end of the tick mark closest to the slider shaft.

yTickPos (USHORT)
Y-coordinate.

Y-coordinate of the point that represents the position of the tick mark. It is the starting position of the tick mark and represents the end of the tick mark closest to the slider shaft.

If NULL is returned in either parameter, an error occurred. The [WinGetLastError](#) function may return the following error:

PMERR_PARAMETER_OUT_OF_RANGE.

SLM_QUERYTICKPOS - Syntax

This message queries for the current position of a tick mark for the primary scale. This represents where the tick mark would be located. The tick mark does not have to have a size (that is, to be visible) to use this message.

```
param1
    USHORT  usTickNum    /* Tick mark location. */

param2
    ULONG    ulReserved  /* Reserved value, should be 0. */

returns
    USHORT  xTickPos     /* X-coordinate. */
    USHORT  yTickPos     /* Y-coordinate. */
```

SLM_QUERYTICKPOS - Remarks

This message could be used to get the position of a tick mark along the slider for use in ownerdraw mode if, for example, you want to place something other than text, such as bit maps or icons, above the tick marks.

SLM_QUERYTICKPOS - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

SLM_QUERYTICKPOS - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

SLM_QUERYTICKSIZE

SLM_QUERYTICKSIZE Field - usTickNum

usTickNum (USHORT)
Tick mark location.

Specifies the tick mark location to query for the size.

SLM_QUERYTICKSIZE Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

SLM_QUERYTICKSIZE Return Value - usTickSize

usTickSize (USHORT)
Tick mark length.

Specifies the length of the tick mark at the position queried, in pixels. If this value is 0, the tick mark is invisible.

If the SLDERR_INVALID_PARAMETERS error is returned, an error occurred. The [WinGetLastError](#) function may return the following error:

PMERR_PARAMETER_OUT_OF_RANGE.

SLM_QUERYTICKSIZE - Parameters

usTickNum (USHORT)
Tick mark location.

Specifies the tick mark location to query for the size.

ulReserved (ULONG)
Reserved value, should be 0.

usTickSize (USHORT)
Tick mark length.

Specifies the length of the tick mark at the position queried, in pixels. If this value is 0, the tick mark is invisible.

If the SLDERR_INVALID_PARAMETERS error is returned, an error occurred. The [WinGetLastError](#) function may return the following error:

PMERR_PARAMETER_OUT_OF_RANGE.

SLM_QUERYTICKSIZE - Syntax

This message queries for the size of a tick mark for the primary scale. All tick marks default to a size of 0 (invisible) if not set by the

application with the SLM_SETTICKSIZE message.

```
param1
    USHORT  usTickNum    /* Tick mark location. */

param2
    ULONG   ulReserved   /* Reserved value, should be 0. */
```

SLM_QUERYTICKSIZE - Remarks

The application uses this message to query a scale along the slider to indicate what tick marks, tick mark sizes, or both are currently set for the slider.

SLM_QUERYTICKSIZE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

SLM_QUERYTICKSIZE - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

SLM_REMOVEDETENT

SLM_REMOVEDETENT Field - ulDetentId

ulDetentId ([ULONG](#))
Detent ID.

Unique detent identifier for the detent that is to be removed from the slider.

SLM_REMOVEDETENT Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

SLM_REMOVEDETENT Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE
Detent was successfully removed.

FALSE
An error occurred. The [WinGetLastError](#) function may return the following error:
PMERR_INVALID_PARAMETERS.

SLM_REMOVEDETENT - Parameters

ulDetentId ([ULONG](#))
Detent ID.

Unique detent identifier for the detent that is to be removed from the slider.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Success indicator.

TRUE
Detent was successfully removed.

FALSE
An error occurred. The [WinGetLastError](#) function may return the following error:
PMERR_INVALID_PARAMETERS.

SLM_REMOVEDETENT - Syntax

This message removes a previously specified detent. A detent is an indicator that represents a predefined value for a quantity and does not have to correspond to an increment of the slider.

```
param1
    ULONG ulDetentId /* Detent ID. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

SLM_REMOVEDETENT - Remarks

The application uses this message to remove detents added previously to the slider to denote values that do not fall along an increment setting.

SLM_REMOVEDETENT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

SLM_REMOVEDETENT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

SLM_SETSCALETEXT

SLM_SETSCALETEXT Field - usTickNum

usTickNum ([USHORT](#))
Tick mark location.

Specifies the tick mark location that is to have the text placed with it.

SLM_SETSCALETEXT Field - pTickText

pTickText ([PSZ](#))

Pointer to the text that is to be drawn at the position specified.

If this value is NULL, no text is drawn.

SLM_SETSCALETEXT Return Value - rc

rc ([BOOL](#))

Success indicator.

TRUE

Text was successfully added to the scale.

FALSE

An error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_HEAP_MAX_SIZE_REACHED
- PMERR_PARAMETER_OUT_OF_RANGE.

SLM_SETSCALETEXT - Parameters

usTickNum ([USHORT](#))

Tick mark location.

Specifies the tick mark location that is to have the text placed with it.

pTickText ([PSZ](#))

Pointer to the text that is to be drawn at the position specified.

If this value is NULL, no text is drawn.

rc ([BOOL](#))

Success indicator.

TRUE

Text was successfully added to the scale.

FALSE

An error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_HEAP_MAX_SIZE_REACHED
- PMERR_PARAMETER_OUT_OF_RANGE.

SLM_SETSCALETEXT - Syntax

This message sets text above a tick mark for the primary scale. A tick mark does not have to be visible to have text set above it. The text is centered on the tick mark.

```
param1
    USHORT  usTickNum  /* Tick mark location. */

param2
    PSZ      pTickText /* Pointer to the text that is to be drawn at the position specified. */
```

SLM_SETSCALETEXT - Remarks

The application uses this message to draw text along the increments of the slider to clarify the magnitude of the range. This text could show the exact value for that tick mark, or could be a general remark, such as low, high, and so forth.

SLM_SETSCALETEXT - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

SLM_SETSCALETEXT - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

SLM_SETSLIDERINFO

SLM_SETSLIDERINFO Field - usInfoType

usInfoType ([USHORT](#))

Component attribute.

Identifies the slider component that is to be modified. Specify one of the following:

SMA_SHAFTDIMENSIONS

Sets the width (for vertical sliders) or height (for horizontal sliders) of the slider shaft.

SMA_SHAFTPOSITION

Sets the x-, y-position of the lower-left corner of the slider shaft in the slider window.

SMA_SLIDERARMDIMENSIONS

Sets the width and height of the slider arm.

SMA_SLIDERARMPOSITION

Sets the position of the slider arm. This value can be specified either as an increment position or a range value.

SLM_SETSLIDERINFO Field - usArmPosType

usArmPosType (USHORT)

Format attribute.

Identifies the format in which the information should be interpreted by the slider if setting the slider arm position is requested. This value is a reserved field for other set requests. The format is one of the following:

SMA_RANGEVALUE

Number of pixels between the home position and the current arm position.

SMA_INCREMENTVALUE

Increment position using the primary scale.

SLM_SETSLIDERINFO Field - ullInfo

ullInfo (ULONG)

New value.

New value to change the slider component to. The format of the information depends on the component being changed and is indicated by the SMA_* message attribute or attributes that are set.

- If the SMA_SHAFTDIMENSIONS attribute is set, the *ullInfo* parameter is as follows:

usShaftBreadth (USHORT)

Width (for vertical sliders) or height (for horizontal sliders) the slider shaft should be set to, in pixels. This is the breadth the shaft should be.

- If the SMA_SHAFTPOSITION attribute is set, the *ullInfo* parameter is as follows:

xShaftCoord (USHORT)

X-coordinate to set the position of the shaft to within the slider window. This value is expressed in window coordinates and represents the lower-left corner of the shaft.

yShaftCoord (USHORT)

Y-coordinate to set the position of the shaft to within the slider window. This value is expressed in window coordinates and represents the lower-left corner of the shaft.

- If the SMA_SLIDERARMDIMENSIONS attribute is set, the *ullInfo* parameter is as follows:

usArmLength (USHORT)

Length of the slider arm, in pixels. This is the width of the arm for horizontal sliders and the height of the arm for vertical sliders.

usArmBreadth (USHORT)

Breadth of the slider arm, in pixels. This is the height of the arm for horizontal sliders and the width of the arm for vertical sliders.

- If the SMA_SLIDERARMPOSITION and SMA_RANGEVALUE attributes are set, the *ullInfo* parameter is as follows:

usArmPos (USHORT)

Number of pixels to be set from home to the slider arm.

- If the SMA_SLIDERARMPOSITION and SMA_INCREMENTVALUE attributes are set, the *ullInfo* parameter is as follows:

usIncrementPos (**USHORT**)

Increment value which corresponds to the position the slider arm should be set to.

SLM_SETSLIDERINFO Return Value - rc

rc (**BOOL**)

Success indicator.

TRUE

Slider component was successfully set.

FALSE

An error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_PARAMETER_OUT_OF_RANGE.

SLM_SETSLIDERINFO - Parameters

usInfoType (**USHORT**)

Component attribute.

Identifies the slider component that is to be modified. Specify one of the following:

SMA_SHAFTDIMENSIONS

Sets the width (for vertical sliders) or height (for horizontal sliders) of the slider shaft.

SMA_SHAFTPOSITION

Sets the x-, y-position of the lower-left corner of the slider shaft in the slider window.

SMA_SLIDERARMDIMENSIONS

Sets the width and height of the slider arm.

SMA_SLIDERARMPOSITION

Sets the position of the slider arm. This value can be specified either as an increment position or a range value.

usArmPosType (**USHORT**)

Format attribute.

Identifies the format in which the information should be interpreted by the slider if setting the slider arm position is requested. This value is a reserved field for other set requests. The format is one of the following:

SMA_RANGEVALUE

Number of pixels between the home position and the current arm position.

SMA_INCREMENTVALUE

Increment position using the primary scale.

ullInfo (**ULONG**)

New value.

New value to change the slider component to. The format of the information depends on the component being changed and is indicated by the SMA_* message attribute or attributes that are set.

- If the SMA_SHAFTDIMENSIONS attribute is set, the *ullInfo* parameter is as follows:

usShaftBreadth (USHORT)

Width (for vertical sliders) or height (for horizontal sliders) the slider shaft should be set to, in pixels. This is the breadth the shaft should be.

- If the SMA_SHAFTPOSITION attribute is set, the *ullInfo* parameter is as follows:

xShaftCoord (USHORT)

X-coordinate to set the position of the shaft to within the slider window. This value is expressed in window coordinates and represents the lower-left corner of the shaft.

yShaftCoord (USHORT)

Y-coordinate to set the position of the shaft to within the slider window. This value is expressed in window coordinates and represents the lower-left corner of the shaft.

- If the SMA_SLIDERARMDIMENSIONS attribute is set, the *ullInfo* parameter is as follows:

usArmLength (USHORT)

Length of the slider arm, in pixels. This is the width of the arm for horizontal sliders and the height of the arm for vertical sliders.

usArmBreadth (USHORT)

Breadth of the slider arm, in pixels. This is the height of the arm for horizontal sliders and the width of the arm for vertical sliders.

- If the SMA_SLIDERARMPOSITION and SMA_RANGEVALUE attributes are set, the *ullInfo* parameter is as follows:

usArmPos (USHORT)

Number of pixels to be set from home to the slider arm.

- If the SMA_SLIDERARMPOSITION and SMA_INCREMENTVALUE attributes are set, the *ullInfo* parameter is as follows:

usIncrementPos (USHORT)

Increment value which corresponds to the position the slider arm should be set to.

rc (BOOL)

Success indicator.

TRUE

Slider component was successfully set.

FALSE

An error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_PARAMETER_OUT_OF_RANGE.

SLM_SETSLIDERINFO - Syntax

This message sets the current position or dimensions of a key component of the slider. The component to be changed is indicated by one parameter and the new value is placed in the other.

```
param1
    USHORT  usInfoType    /* Component attribute. */
    USHORT  usArmPosType  /* Format attribute. */

param2
    ULONG   ulInfo        /* New value. */
```

SLM_SETSLIDERINFO - Remarks

The application uses this message to customize the slider for a specific use. In setting the shaft dimensions, only the breadth of the slider can be set. The length of the shaft is always determined by the number of increments and the spacing between increments, both of which are set for the primary scale when the slider is created.

Positioning of the shaft within the slider window could be used by applications that cannot use the default positioning provided by the slider control.

Setting of the slider arm dimensions could be used by applications that need a larger slider arm, such as touch screen applications.

Setting the slider arm position can be used to:

- Set the initial value of the slider before it becomes visible.
- Change the value when it is tied to another control, such as an entry field.
- Show the value of a quantity when the slider is being used to monitor an event, such as a read-only slider being used as a progress indicator.

SLM_SETSLIDERINFO - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

SLM_SETSLIDERINFO - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

SLM_SETTICKSIZE

SLM_SETTICKSIZE Field - usTickNum

usTickNum ([USHORT](#))

Tick mark location.

Tick mark location whose size is to be changed. If the SMA_SETALLTICKS attribute is specified for this parameter, all tick marks on the primary scale are set to the size specified.

SLM_SETTICKSIZE Field - usTickSize

usTickSize ([USHORT](#))
Tick mark length.

Length of the tick mark, in pixels. If set to 0, the tick mark will not be drawn.

SLM_SETTICKSIZE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

SLM_SETTICKSIZE Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE
Tick mark position was successfully set.

FALSE
An error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_HEAP_MAX_SIZE_REACHED
- PMERR_PARAMETER_OUT_OF_RANGE.

SLM_SETTICKSIZE - Parameters

usTickNum ([USHORT](#))
Tick mark location.

Tick mark location whose size is to be changed. If the SMA_SETALLTICKS attribute is specified for this parameter, all tick marks on the primary scale are set to the size specified.

usTickSize ([USHORT](#))
Tick mark length.

Length of the tick mark, in pixels. If set to 0, the tick mark will not be drawn.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Success indicator.

TRUE
Tick mark position was successfully set.

FALSE

An error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_HEAP_MAX_SIZE_REACHED
- PMERR_PARAMETER_OUT_OF_RANGE.

SLM_SETTICKSIZE - Syntax

This message sets the size of a tick mark for the primary scale. All tick marks are initially set to a size of 0 (invisible). Each tick mark along a scale can be set to the size desired.

```
param1
    USHORT  usTickNum    /* Tick mark location. */
    USHORT  usTickSize   /* Tick mark length. */

param2
    ULONG   ulReserved   /* Reserved value, should be 0. */
```

SLM_SETTICKSIZE - Remarks

The application uses this message to draw a scale along the slider to indicate value positions in relation to the slider arm. The application can set varying lengths for different increments of the slider to help the user understand the magnitude of the value being set.

SLM_SETTICKSIZE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

SLM_SETTICKSIZE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_CHAR (in Slider Controls)

WM_CHAR (in Slider Controls) - Syntax

For the cause of this message, see [WM_CHAR](#).

For a description of the parameters, see [WM_CHAR](#).

WM_CHAR (in Slider Controls) - Remarks

The slider control window procedure responds to this message by sending it to its owner if it has not processed the key stroke. This is the most common means by which the input focus is switched around the various controls in a dialog box.

The keystrokes processed by a linear slider control are:

Down Arrow	Moves the slider arm down one increment. When the slider arm reaches the bottom of the slider shaft or when a horizontal slider is being used, the Down Arrow key has no effect.
Up Arrow	Moves the slider arm up one increment. When the slider arm reaches the top of the slider shaft or when a horizontal slider is being used, the Up Arrow key has no effect.
Left Arrow	Moves the slider arm left one increment. When the slider arm reaches the leftmost edge or when a vertical slider is being used, the Left Arrow key has no effect.
Right Arrow	Moves the slider arm right one increment. When the slider arm reaches the rightmost edge or when a vertical slider is being used, the Right Arrow key has no effect.
Shift+Down Arrow	Moves the slider arm to the next detent below the current position. If there are no more detents or if a horizontal slider is being used, the Shift+Down Arrow key combination has no effect.
Shift+Up Arrow	Moves the slider arm to the next detent above the current position. If there are no more detents or if a horizontal slider is being used, the Shift+Up Arrow key combination has no effect.
Shift+Left Arrow	Moves the slider arm to the next detent left of the current position. If there are no more detents or if a vertical slider is being used, the Shift+Left Arrow key combination has no effect.
Shift+Right Arrow	Moves the slider arm to the next detent right of the current position. If there are no more detents or if a vertical slider is being used, the Shift+Right Arrow key combination has no effect.
Home, Ctrl+Home	Moves the slider arm to the home position of the slider. Pressing the Home key or the Ctrl+Home key combination when the slider arm is at the home position has no effect. The default home position for a slider is the leftmost edge for horizontal sliders and the bottom edge for vertical sliders.
End, Ctrl+End	Moves the slider arm to the end position of the slider. Pressing the End key or the Ctrl+End key combination when the slider arm is at the end position has no effect. The default end position for a slider is the rightmost edge for horizontal sliders and the top edge for vertical sliders.

A circular slider control only processes left and right arrow keystrokes. These keys move the slider arm one increment to the left or right.

WM_CHAR (in Slider Controls) - Default Processing

For a description of the default processing, see [WM_CHAR](#).

WM_CHAR (in Slider Controls) - Topics

Select an item:

[Syntax](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

WM_PRESPARAMCHANGED (in Slider Controls)

WM_PRESPARAMCHANGED (in Slider Controls) Field - attrtype

attrtype ([ULONG](#))

Attribute type.

Presentation parameter attribute identity. The following presentation parameters are initialized by the slider control. The initial value of each is shown in the following list:

PP_FOREGROUND_COLOR or PP_FOREGROUND_COLORINDEX

Item foreground color; used when displaying text and bit maps. This color is initialized to SYSCLR_WINDOWTEXT.

PP_BACKGROUND_COLOR or PP_BACKGROUND_COLORINDEX

Slider background color; used for entire control as the background. This color is initialized to SYSCLR_WINDOW.

WM_PRESPARAMCHANGED (in Slider Controls) Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_PRESPARAMCHANGED (in Slider Controls) Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, must be 0.

WM_PRESPARAMCHANGED (in Slider Controls) - Parameters

attrtype ([ULONG](#))
Attribute type.

Presentation parameter attribute identity. The following presentation parameters are initialized by the slider control. The initial value of each is shown in the following list:

PP_FOREGROUND_COLOR or PP_FOREGROUND_COLOR_INDEX
Item foreground color; used when displaying text and bit maps. This color is initialized to SYSCLR_WINDOWTEXT.

PP_BACKGROUND_COLOR or PP_BACKGROUND_COLOR_INDEX
Slider background color; used for entire control as the background. This color is initialized to SYSCLR_WINDOW.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, must be 0.

WM_PRESPARAMCHANGED (in Slider Controls) - Syntax

For the cause of this message, see [WM_PRESPARAMCHANGED](#).

```
param1
    ULONG attrtype    /* Attribute type. */

param2
    ULONG ulReserved  /* Reserved value, should be 0. */
```

WM_PRESPARAMCHANGED (in Slider Controls) - Remarks

The application uses this message to notify the slider that a given inherited presentation parameter has changed.

WM_PRESPARAMCHANGED (in Slider Controls) - Default Processing

For a description of the default processing, see [WM_PRESPARAMCHANGED](#).

WM_PRESPARAMCHANGED (in Slider Controls) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_QUERYWINDOWPARAMS (in Slider Controls)

WM_QUERYWINDOWPARAMS (in Slider Controls) Field - pwndparams

pwndparams ([PWNDPARAMS](#))

Pointer to a [WNDPARAMS](#) window parameter structure.

This structure contains:

status ([USHORT](#))

Window parameter selection.

Identifies the window parameters that are to be set or queried. Valid values for the slider control are:

WPM_CBCTLDATA

Window control data length.

WPM_CTLDATA

Window control data.

The flags in the *status* field are cleared as each item is processed. If the call is successful, the *status* field is 0. If any item has not been processed, the flag for that item is still set.

length ([USHORT](#))

Length of the window text.

text ([PSZ](#))

Window text.

presparamslength ([USHORT](#))

Length of presentation parameters.

presparams ([PVOID](#))

Presentation parameters.

ctldatalength ([USHORT](#))

Length of window class-specific data.

ctldata ([PVOID](#))

Window class-specific data.

WM_QUERYWINDOWPARAMS (in Slider Controls) Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_QUERYWINDOWPARAMS (in Slider Controls) Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

WM_QUERYWINDOWPARAMS (in Slider Controls) - Parameters

pwndparams (PWNDPARAMS)
Pointer to a WNDPARAMS window parameter structure.

This structure contains:

status (USHORT)
Window parameter selection.

Identifies the window parameters that are to be set or queried. Valid values for the slider control are:

WPM_CBCTLDATA	Window control data length.
WPM_CTLDATA	Window control data.

The flags in the *status* field are cleared as each item is processed. If the call is successful, the *status* field is 0. If any item has not been processed, the flag for that item is still set.

length (USHORT)
Length of the window text.

text (PSZ)
Window text.

presparamslength (USHORT)
Length of presentation parameters.

presparams (PVOID)
Presentation parameters.

ctldatalength ([USHORT](#))
Length of window class-specific data.

ctldata ([PVOID](#))
Window class-specific data.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Success indicator.

TRUE
Successful completion.

FALSE
Error occurred.

WM_QUERYWINDOWPARAMS (in Slider Controls) - Syntax

For the cause of this message, see [WM_QUERYWINDOWPARAMS](#).

```
param1  
    PWNDPARAMS    pwndparams /* Pointer to a WNDPARAMS window parameter structure. */  
  
param2  
    ULONG          ulReserved /* Reserved value, should be 0. */
```

WM_QUERYWINDOWPARAMS (in Slider Controls) - Remarks

The slider control window procedure responds to this message by returning the information in the buffer provided. If this message is sent to a slider window of another process, the information in, or identified by, the value of the *pwndparams* field must be in memory shared by both processes.

WM_QUERYWINDOWPARAMS (in Slider Controls) - Default Processing

For a description of the default processing, see [WM_QUERYWINDOWPARAMS](#).

WM_QUERYWINDOWPARAMS (in Slider Controls) - Topics

Select an item:

[Syntax](#)

[Parameters](#)

WM_SETWINDOWPARAMS (in Slider Controls)

WM_SETWINDOWPARAMS (in Slider Controls) Field - pwndparams

pwndparams (PWNDPARAMS)	Pointer to a WNDPARAMS window parameter structure.		
This structure contains:			
<i>status</i> (USHORT)	Window parameter selection. Identifies the window parameters that are to be set or queried. The valid value for the slider control is: <table><tr><td>WPM_CTLDATA</td><td>Window control data.</td></tr></table> The flags in the <i>status</i> field are cleared as each item is processed. If the call is successful, the <i>status</i> field is 0. If any item has not been processed, the flag for that item is still set.	WPM_CTLDATA	Window control data.
WPM_CTLDATA	Window control data.		
<i>length</i> (USHORT)	Length of the window text.		
<i>text</i> (PSZ)	Window text.		
<i>presparamslength</i> (USHORT)	Length of presentation parameters.		
<i>presparams</i> (PVOID)	Presentation parameters.		
<i>ctldatalength</i> (USHORT)	Length of window class-specific data.		
<i>ctldata</i> (PVOID)	Window class-specific data.		

WM_SETWINDOWPARAMS (in Slider Controls) Field - ulReserved

ulReserved (ULONG)	Reserved value, should be 0.
---	------------------------------

WM_SETWINDOWPARAMS (in Slider Controls) Return Value - rc

rc (BOOL)

Success indicator.

TRUE

Successful operation

FALSE

Error occurred.

WM_SETWINDOWPARAMS (in Slider Controls) - Parameters

pwndparams (PWNDPARAMS)

Pointer to a WNDPARAMS window parameter structure.

This structure contains:

status (USHORT)

Window parameter selection.

Identifies the window parameters that are to be set or queried. The valid value for the slider control is:

WPM_CTLDATA

Window control data.

The flags in the *status* field are cleared as each item is processed. If the call is successful, the *status* field is 0. If any item has not been processed, the flag for that item is still set.

length (USHORT)

Length of the window text.

text (PSZ)

Window text.

presparamslength (USHORT)

Length of presentation parameters.

presparams (PVOID)

Presentation parameters.

ctldatalength (USHORT)

Length of window class-specific data.

ctldata (PVOID)

Window class-specific data.

ulReserved (ULONG)

Reserved value, should be 0.

rc (BOOL)

Success indicator.

TRUE

Successful operation

FALSE

Error occurred.

WM_SETWINDOWPARAMS (in Slider Controls) - Syntax

For the cause of this message, see [WM_SETWINDOWPARAMS](#).

```
param1
    PWNDPARAMS pwndparams /* Pointer to a WNDPARAMS window parameter structure. */

param2
    ULONG      ulReserved /* Reserved value, should be 0. */
```

WM_SETWINDOWPARAMS (in Slider Controls) - Remarks

If this message is sent to a slider window of another process, the information in, or identified by, the value of the *pwndparams* field must be in memory shared by both processes.

WM_SETWINDOWPARAMS (in Slider Controls) - Default Processing

For a description of the default processing, see [WM_SETWINDOWPARAMS](#).

WM_SETWINDOWPARAMS (in Slider Controls) - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

Spin Button Control Window Processing

This system-provided window procedure processes the actions on a spin button control (WC_SPINBUTTON).

Purpose

A spin button control (WC_SPINBUTTON window class) is a visual component whose specific purpose is to give users quick access to a finite set of data. The spin button allows users to select from a scrollable ring of choices. Since users can see only one item at a time, the

spin button control should be used only with data that is intuitively related, such as a list of months of the year, or an alphabetic list of cities or states.

A spin button consists of at least one spin field that is a single-line entry field (SLE), and up and down arrows that are stacked on top of one another. These arrows are positioned at the right of the SLE.

You can create multifield spin buttons for those applications in which users must select more than one value. For example, in setting a date the spin button control can provide individual fields for setting the month, day, and year. The first spin field in the spin button could contain a list of months, the second spin field could contain a list of numbers and the third spin field could contain a list of years.

Spin Button Control Styles

Create a spin button using the style bits listed below. These styles can be joined together by using logical ORs (|).

- Specify one of the following to determine whether a spin field will be a master or a servant. If neither is specified, SPBS_SERVANT is the default.

SPBS_MASTER

The spin button component consists of at least one single line entry field (SLE), or spin field, and two arrows, the Up Arrow and the Down Arrow. When a spin button contains more than one spin field, the master component contains the spin arrows. If the component contains only one spin field, it should be a master.

SPBS_SERVANT

You can create a multifield spin button by spinning servants from the master.
- Specify one of the following to determine the type of characters allowed in the spin field:

SPBS_ALLCHARACTERS

Any character can be typed in the spin field. This is the default.

SPBS_NUMERICONLY

Only the digits 0-9 and the minus sign (-) can be typed in the spin field.

SPBS_READONLY

Nothing can be typed in the spin field.
- Specify one of the following to determine how the text is to be presented in the spin field:

SPBS_JUSTLEFT

Left-justify the text. This is the default.

SPBS_JUSTRIGHT

Right-justify the text.

SPBS_JUSTCENTER

Center the text.
- Specify the following when you do not want a border around the spin button:

SPBS_NOBORDER

Suppresses drawing a border.
- Specify the following to increase the spin speed:

SPBS_FASTSPIN

Enables the spin button to increase the spin speed with time. The speed doubles every two seconds.

Note: The spin button skips information when this option is specified. Do not use SPBS_FASTSPIN if the application requires that this field be checked each time a spin up or spin down occurs. Do not specify this option on a master component that has servants spun from it.
- Specify the following to pad numeric fields with 0s. This is useful when the spin field contains values that represent time or money.

SPBS_PADWITHZEROS

The output number is padded at the front between the first non-zero digit and the field width, or 11 characters, whichever is the lesser. The negative sign, if there is one, is retained. The maximum number of characters required to display a **LONG** number is 11.

Spin Button Control Data

See [SPBCDATA](#)

Spin Button Control Notification Message

This message is initiated by the spin button control window to notify its owner of significant events.

WM_CONTROL (in Spin Button Controls)

WM_CONTROL (in Spin Button Controls) Field - id

id ([USHORT](#))
Identity of the spin button component window.

WM_CONTROL (in Spin Button Controls) Field - notifycode

- notifycode** ([USHORT](#))
Notification code.
- SPBN_UPARROW
Tells the application that the Up Arrow was clicked on, or the Up Arrow key was pressed.
 - SPBN_DOWNARROW
Tells the application that the Down Arrow was clicked on, or the Down Arrow key was pressed.
 - SPBN_SETFOCUS
Tells the application which spin field was selected.
 - SPBN_KILLFOCUS
Tells the application when the spin field loses focus.
 - SPBN_ENDSPIN
Tells the application that the user released the select button or one of the arrow keys while spinning a button.
 - SPBN_CHANGE
Tells the application that the contents of the spin field changed.

WM_CONTROL (in Spin Button Controls) Field - hwnd

hwnd (HWND)

Window handle.

The interpretation of this handle is dependent upon the following notification codes:

- SPBN_UPARROW, SPBN_DOWNARROW, and SPBN_ENDSPIN.

The *param2* parameter is the handle to the currently selected spin field in a particular master-servant setup. If either the Up or Down Arrow is clicked on and none of a spin button's servants are currently selected, the master will return a handle to itself.

- SPBN_SETFOCUS

The *param2* parameter is the handle of the currently selected spin field.

This message tells the application which spin field is selected.

- SPBN_KILLFOCUS

The *param2* parameter is NULLHANDLE if the spin field loses focus or no spin field is currently selected.

This message tells the application when a spin field loses focus.

Note: Both SPBN_KILLFOCUS and SPBN_SETFOCUS are set independently. You must check this message only when the application does not specify a master-servant relationship.

- SPBN_CHANGE

The *param2* parameter is the handle of the spin button in which the spin field text changed.

WM_CONTROL (in Spin Button Controls) Return Value - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

WM_CONTROL (in Spin Button Controls) - Parameters

id (USHORT)

Identity of the spin button component window.

notifycode (USHORT)

Notification code.

SPBN_UPARROW

Tells the application that the Up Arrow was clicked on, or the Up Arrow key was pressed.

SPBN_DOWNARROW

Tells the application that the Down Arrow was clicked on, or the Down Arrow key was pressed.

SPBN_SETFOCUS

Tells the application which spin field was selected.

SPBN_KILLFOCUS

Tells the application when the spin field loses focus.

SPBN_ENDSPIN

Tells the application that the user released the select button or one of the arrow keys while spinning a button.

SPBN_CHANGE

Tells the application that the contents of the spin field changed.

hwnd ([HWND](#))

Window handle.

The interpretation of this handle is dependent upon the following notification codes:

- SPBN_UPARROW, SPBN_DOWNARROW, and SPBN_ENDSPIN.

The *param2* parameter is the handle to the currently selected spin field in a particular master-servant setup. If either the Up or Down Arrow is clicked on and none of a spin button's servants are currently selected, the master will return a handle to itself.

- SPBN_SETFOCUS

The *param2* parameter is the handle of the currently selected spin field.

This message tells the application which spin field is selected.

- SPBN_KILLFOCUS

The *param2* parameter is NULLHANDLE if the spin field loses focus or no spin field is currently selected.

This message tells the application when a spin field loses focus.

Note: Both SPBN_KILLFOCUS and SPBN_SETFOCUS are set independently. You must check this message only when the application does not specify a master-servant relationship.

- SPBN_CHANGE

The *param2* parameter is the handle of the spin button in which the spin field text changed.

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_CONTROL (in Spin Button Controls) - Syntax

For the cause of this message, see [WM_CONTROL](#).

```
param1
    USHORT id          /* Identity of the spin button component window. */
    USHORT notifycode  /* Notification code. */

param2
    HWND hwnd         /* Window handle. */
```

WM_CONTROL (in Spin Button Controls) - Remarks

This message is sent when, as specified by *notifycode*, the spin button component must tell its owner of a significant event.

WM_CONTROL (in Spin Button Controls) - Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return 0.

WM_CONTROL (in Spin Button Controls) - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

Spin Button Control Window Messages

This section describes the spin button control window procedure actions on receiving the following messages.

SPBM_OVERRIDESETLIMITS

SPBM_OVERRIDESETLIMITS Field - IUpLimit

IUpLimit ([LONG](#))
Upper limit.

SPBM_OVERRIDESETLIMITS Field - ILowLimit

ILowLimit ([LONG](#))
Lower limit.

SPBM_OVERRIDESETLIMITS Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

SPBM_OVERRIDESETLIMITS - Parameters

IUpLimit ([LONG](#))
Upper limit.

ILowLimit ([LONG](#))
Lower limit.

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

SPBM_OVERRIDESETLIMITS - Syntax

This message causes the component to set or reset numeric limits.

```
param1
    LONG lUpLimit /* Upper limit. */

param2
    LONG lLowLimit /* Lower limit. */
```

SPBM_OVERRIDESETLIMITS - Remarks

The application sends this message to the component to set or reset numeric limits.

This message is functionally identical to [SPBM_SETLIMITS](#), except that the current value of the spin button does not change if it is out of range.

When the upper limit is less than the lower limit, FALSE is returned.

SPBM_OVERRIDESETLIMITS - Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return FALSE.

SPBM_OVERRIDESETLIMITS - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

SPBM_QUERYLIMITS

SPBM_QUERYLIMITS Field - pUpLimit

pUpLimit ([PLONG](#))
Pointer to a [LONG](#) that will receive the returned upper limit.

SPBM_QUERYLIMITS Field - pLowLimit

pLowLimit ([PLONG](#))
Pointer to a [LONG](#) that will receive the returned lower limit.

SPBM_QUERYLIMITS Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SPBM_QUERYLIMITS - Parameters

plUpLimit (PLONG)

Pointer to a [LONG](#) that will receive the returned upper limit.

plLowLimit (PLONG)

Pointer to a [LONG](#) that will receive the returned lower limit.

rc (BOOL)

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

SPBM_QUERYLIMITS - Syntax

This message enables an application to query the limits of a numeric spin field.

```
param1
    PLONG plUpLimit    /* Pointer to a LONG that will receive the returned upper limit. */
param2
    PLONG plLowLimit   /* Pointer to a LONG that will receive the returned lower limit. */
```

SPBM_QUERYLIMITS - Remarks

The application sends this message to the component to determine the limits of a numeric spin field.

When the spin button has no data, or when it is spinning an array, FALSE is returned.

SPBM_QUERYLIMITS - Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return FALSE

SPBM_QUERYLIMITS - Topics

Select an item:

[Syntax](#)

[Parameters](#)

SPBM_QUERYVALUE

SPBM_QUERYVALUE Field - pStorage

pStorage ([PVOID](#))

- Place for returned value.
- A place for the returned value. This value is either the address of a string or the address of a long variable.
- If the *usBufSize* is 0, *param1* is assumed to be an address of a long variable.
- If *param1* is Other, it is assumed to be an address of a string.
- NULL
 - Causes the spin button to process the reset or update as specified, but it will not try to return a value to the application.
- Other
 - The address where the value is returned.

SPBM_QUERYVALUE Field - usBufSize

usBufSize ([USHORT](#))

- Buffer size.
- If *usBufSize* is too small to return all of the text, the spin button returns as much of the text as it can.
- 0
 - The spin button assumes that *param1* is the address of a long variable. If the data in the spin button is spinning between an upper and lower limit, the current value is passed back in the variable.
 - If the data in the spin button is in an array, the index of the current array value (or last valid value) is passed back in the variable.
- Other
 - The spin button assumes that *param1* is the address of a string. The information passed back in the string is dependent upon the flags in the *usValue* parameter.

SPBM_QUERYVALUE Field - usValue

usValue (USHORT)

Update/reset value.

Controls how the spin field is updated.

SPBQ_UPDATEIFVALID

Update the contents of the spin field if the value is valid. This is the default.

Specifying this flag on a query will *not* update the contents of the spin field if it is *exactly* the same as an item in the spin button list.

If an item in the list is Monday, specifying SPBQ_UPDATEIFVALID updates the spin field contents when MONDAY, monday, or mONDAY are typed, but not when Monday is typed. This prevents recursion if the application checks for the validity each time a SPBN_CHANGE message is sent from the component.

SPBQ_ALWAYSUPDATE

Update the contents of the spin field if the value is valid. Reset the contents of the spin field to the last valid value if the field contains data that is not valid.

If the spin button is spinning numbers between an upper and a lower limit, and the content of the spin field is a valid number that is out of range, the spin button does not reset itself to the last valid value. It sets the current position at the upper limit when the out-of-range number specified is above the upper limit. It sets the current position at the lower limit when the out-of-range number is below the lower limit.

When the current value is changed, the return of the query message is still FALSE.

SPBQ_DONOTUPDATE

Do not update the contents of the spin field, even if the value is valid.

SPBM_QUERYVALUE Return Value - rc

rc (BOOL)

Success indicator.

TRUE

Successful completion.

FALSE

Error occurred.

SPBM_QUERYVALUE - Parameters

pStorage (PVOID)

Place for returned value.

A place for the returned value. This value is either the address of a string or the address of a long variable.

If the *usBufSize* is 0, *param1* is assumed to be an address of a long variable.

If *param1* is Other, it is assumed to be an address of a string.

NULL

Causes the spin button to process the reset or update as specified, but it will not try to return a value to the application.

Other

The address where the value is returned.

usBufSize (USHORT)

Buffer size.

If *usBufSize* is too small to return all of the text, the spin button returns as much of the text as it can.

0

The spin button assumes that *param1* is the address of a long variable. If the data in the spin button is spinning between an upper and lower limit, the current value is passed back in the variable.

If the data in the spin button is in an array, the index of the current array value (or last valid value) is passed back in the variable.

Other

The spin button assumes that *param1* is the address of a string. The information passed back in the string is dependent upon the flags in the *usValue* parameter.

usValue (USHORT)

Update/reset value.

Controls how the spin field is updated.

SPBQ_UPDATEIFVALID

Update the contents of the spin field if the value is valid. This is the default.

Specifying this flag on a query will *not* update the contents of the spin field if it is *exactly* the same as an item in the spin button list.

If an item in the list is Monday, specifying SPBQ_UPDATEIFVALID updates the spin field contents when MONDAY, monday, or mONDAY are typed, but not when Monday is typed. This prevents recursion if the application checks for the validity each time a SPBN_CHANGE message is sent from the component.

SPBQ_ALWAYSUPDATE

Update the contents of the spin field if the value is valid. Reset the contents of the spin field to the last valid value if the field contains data that is not valid.

If the spin button is spinning numbers between an upper and a lower limit, and the content of the spin field is a valid number that is out of range, the spin button does not reset itself to the last valid value. It sets the current position at the upper limit when the out-of-range number specified is above the upper limit. It sets the current position at the lower limit when the out-of-range number is below the lower limit.

When the current value is changed, the return of the query message is still FALSE.

SPBQ_DONOTUPDATE

Do not update the contents of the spin field, even if the value is valid.

rc (BOOL)

Success indicator.

TRUE

Successful completion.

FALSE

Error occurred.

SPBM_QUERYVALUE - Syntax

This message causes the component to show the value in the spin field.

```
param1
    PVOID    pStorage    /* Place for returned value. */

param2
    USHORT   usBufSize    /* Buffer size. */
    USHORT   usValue      /* Update/reset value. */
```

SPBM_QUERYVALUE - Remarks

The application sends this message to the component to determine what value is in the spin field. The application sets up a field for the component to deposit the value, and sets a flag to determine what the function does when the value matches or does not match the given spin-list values.

TRUE is returned when a matched value is found, or the data is in the range.

FALSE is returned when no match is found, the value is out of range, or no spin data exists.

SPBM_QUERYVALUE - Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return FALSE.

SPBM_QUERYVALUE - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

SPBM_SETARRAY

SPBM_SETARRAY Field - pStrl

pStrl ([PSZ](#))

Pointer to the new array of values.

SPBM_SETARRAY Field - usItems

usItems ([USHORT](#))

Number of items in the array.

SPBM_SETARRAY Return Value - rc

rc (BOOL)

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

SPBM_SETARRAY - Parameters

pStr1 (PSZ)

Pointer to the new array of values.

usItems (USHORT)

Number of items in the array.

rc (BOOL)

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

SPBM_SETARRAY - Syntax

This message causes the component to set or reset the array of data.

```
param1
    PSZ    pStr1    /* Pointer to the new array of values. */
param2
    USHORT usItems /* Number of items in the array. */
```

SPBM_SETARRAY - Remarks

The application sends this message to the component to set or reset the array of data.

The component tries to leave the current value unchanged. However, if the current value is out of range for the new array, it is moved to the

closest extreme. Thus, if the current value is less than 0, it is moved to 0. If the current value is greater than the previous value, it is set to the previous value.

If the data exceeds 64KB, or if *param1* or *param2* equal 0, FALSE is returned.

SPBM_SETARRAY - Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return FALSE.

SPBM_SETARRAY - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)
-

SPBM_SETCURRENTVALUE

SPBM_SETCURRENTVALUE Field - IValue

- IValue** ([LONG](#))
- Array value or index.
 - Current value or index of array.
-

SPBM_SETCURRENTVALUE Field - ulReserved

- ulReserved** ([ULONG](#))
- Reserved value, should be 0.
-

SPBM_SETCURRENTVALUE Return Value - rc

rc (BOOL)
Success indicator.

TRUE Successful completion

FALSE Error occurred.

SPBM_SETCURRENTVALUE - Parameters

lValue (LONG)
Array value or index.

Current value or index of array.

ulReserved (ULONG)
Reserved value, should be 0.

rc (BOOL)
Success indicator.

TRUE Successful completion

FALSE Error occurred.

SPBM_SETCURRENTVALUE - Syntax

This message causes the component to set or reset the current numeric value or array index.

```
param1
    LONG    lValue        /* Array value or index. */
param2
    ULONG    ulReserved    /* Reserved value, should be 0. */
```

SPBM_SETCURRENTVALUE - Remarks

The application sends this message to the component to set or reset the current numeric value or array index.

FALSE is returned when the value is out of range or there is no spin data.

SPBM_SETCURRENTVALUE - Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return FALSE.

SPBM_SETCURRENTVALUE - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

SPBM_SETLIMITS

SPBM_SETLIMITS Field - IUpLimit

IUpLimit (LONG)
Upper limit.

SPBM_SETLIMITS Field - ILowLimit

ILowLimit (LONG)
Lower limit.

SPBM_SETLIMITS Parameter - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SPBM_SETLIMITS - Parameters

IUpLimit (**LONG**)
Upper limit.

ILowLimit (**LONG**)
Lower limit.

rc (**BOOL**)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SPBM_SETLIMITS - Syntax

This message causes the component to set or reset numeric limits.

```
param1
    LONG  lUpLimit    /* Upper limit. */

param2
    LONG  lLowLimit   /* Lower limit. */
```

SPBM_SETLIMITS - Remarks

The application sends this message to the component to set or reset numeric limits. The component sets the current value to the content in the spin field when it is a valid number. When the current value is out of the range of the limits, it is moved to the nearest limit, upper or lower.

If the upper limit is less than the lower limit, FALSE is returned.

SPBM_SETLIMITS - Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return FALSE.

SPBM_SETLIMITS - Topics

Select an item:

SPBM_SETMASTER

SPBM_SETMASTER Field - hwnd

hwnd (HWND)
Handle of master component.

SPBM_SETMASTER Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

SPBM_SETMASTER Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SPBM_SETMASTER - Parameters

hwnd (HWND)
Handle of master component.

ulReserved (ULONG)
Reserved value, should be 0.

rc ([BOOL](#))

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

SPBM_SETMASTER - Syntax

This message causes the component to identify its master.

```
param1
    HWND    hwnd           /* Handle of master component. */

param2
    ULONG    ulReserved    /* Reserved value, should be 0. */
```

SPBM_SETMASTER - Remarks

The application sends this message to the component to tell a component who its master is.

When the application wants to take control of the spin button, it must set the *param1* of each spin button to NULLHANDLE. This must be done, for example, when a spin button with a non-contiguous list of spin values is created (2, 4, 6, 8, 10...). When the *param1* of a spin button is NULLHANDLE, the spin button does not perform the following default functions:

- Spin up or down on its own when the Up or Down Arrow key is pressed.
- Spin up or down when the Up or Down Arrow of the master is pressed.
- A master does not take the focus when its arrows are pressed and none of its servants have focus.
- The spin button does not send itself an [SPBM_QUERYVALUE](#) message with the SPBQ_ALWAYSUPDATE flag to update the current value when an [SPBM_SPINUP](#) or [SPBM_SPINDOWN](#) message is received.
- The spin button does not fast spin.

SPBM_SETMASTER - Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return FALSE.

SPBM_SETMASTER - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

SPBM_SETTEXTLIMIT

SPBM_SETTEXTLIMIT Field - usLimit

usLimit ([USHORT](#))
Character limit.

Number of characters to allow.

SPBM_SETTEXTLIMIT Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

SPBM_SETTEXTLIMIT Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE Successful completion
FALSE Error occurred.

SPBM_SETTEXTLIMIT - Parameters

usLimit ([USHORT](#))
Character limit.

Number of characters to allow.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

SPBM_SETTEXTLIMIT - Syntax

This message sets the maximum number of characters allowed in a spin field.

```
param1
    USHORT  usLimit      /* Character limit. */

param2
    ULONG   ulReserved   /* Reserved value, should be 0. */
```

SPBM_SETTEXTLIMIT - Remarks

The application sends this message to set the maximum number of characters allowed in the spin field. The size limit of the spin field is 255 characters. This is the default.

When the size exceeds 255 characters, FALSE is returned,

SPBM_SETTEXTLIMIT - Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return FALSE.

SPBM_SETTEXTLIMIT - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

SPBM_SPINDOWN

SPBM_SPINDOWN Field - ullItem

ullItem (ULONG)
Number of values to spin down.

SPBM_SPINDOWN Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

SPBM_SPINDOWN Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SPBM_SPINDOWN - Parameters

ullItem (ULONG)
Number of values to spin down.

ulReserved (ULONG)
Reserved value, should be 0.

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SPBM_SPINDOWN - Syntax

This message causes the component to show the previous value (spin backward).

```
param1
    ULONG  ulItem      /* Number of values to spin down. */

param2
    ULONG  ulReserved  /* Reserved value, should be 0. */
```

SPBM_SPINDOWN - Remarks

The application sends this message to the component when it wants the previous value shown (spin backward).
When there is no data to spin, FALSE is returned.

SPBM_SPINDOWN - Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return FALSE.

SPBM_SPINDOWN - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

SPBM_SPINUP

SPBM_SPINUP Field - ullItem

ullItem ([ULONG](#))
Number of values to spin up.

SPBM_SPINUP Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

SPBM_SPINUP Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SPBM_SPINUP - Parameters

ulItem (ULONG)
Number of values to spin up.

ulReserved (ULONG)
Reserved value, should be 0.

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

SPBM_SPINUP - Syntax

This message causes the component to show the next value (spin forward).

```
param1
    ULONG ulItem      /* Number of values to spin up. */

param2
    ULONG ulReserved  /* Reserved value, should be 0. */
```

SPBM_SPINUP - Remarks

The application sends this message to the component when it wants the next value shown (spin forward).

When there is no data to spin, FALSE is returned.

SPBM_SPINUP - Default Processing

The default window procedure does not expect to receive this message and takes no action other than to return FALSE.

SPBM_SPINUP - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

Static Control Window Processing

This system-provided window procedure processes the actions on a static control (WC_STATIC).

Purpose

Static controls are simple text fields, bit maps, icons, and boxes that can be used to label or box other controls. Static controls do not accept user input, nor do they send notification messages to their owner.

Static Control Data

None.

Static Control Styles

These static control styles are available:

SS_TEXT

Creates a box with formatted text. The text is formatted before it is displayed according to the setting of these text drawing-style flags:

DT_LEFT
DT_CENTER
DT_RIGHT

Left-justified text
Centered text
Right-justified text

ORed with one of:

DT_TOP	Text is aligned to top of window
DT_VCENTER	Text is aligned vertically in center of window
DT_BOTTOM	Text is aligned to bottom of window

The following text drawing style can also be ORed, but only if DT_TOP and DT_LEFT are also specified:

DT_WORDBREAK	Text is multi-line with word-wrapping at ends of lines.
--------------	---

Note: For "static" text that can be selected, a Button Control with a style of BS_NOBORDER can be used.

SS_GROUPBOX

A group box static control is a box that has an identifying text string in its upper left corner. Group boxes are used to collect a group of radio buttons or other controls into a single unit.

SS_ICON

Draws an icon. The text of the static control is a string that is used to derive the resource ID from which the icon is loaded. The format of the string is:

- The first byte is 0xFF, the second byte is the low byte of the resource ID, and the third byte is the high byte of the resource ID.
- The first character is "#"; subsequent characters make up the decimal text representation of the resource ID. This format can be used for specifying a system icon in a resource file. The decimal string is the value of the appropriate SPTR_* constant

If the string is empty or does not follow the format above, no resource is loaded.

The resource is assumed to reside in the resource file of the current process.

This control is resized to the size of the icon.

SS_SYSICON

This style is the same as SS_ICON except that the icon ID is specified as one of the system pointer ID values (SPTR_* values) rather than a resource ID. This style provides a convenient way to include system icons in application dialog boxes.

SS_BITMAP

Draws a bit map. The text of the static control names the bit-map resource, as for SS_ICON.

SS_FGNDRECT

Creates a rectangle filled with the color of the foreground.

SS_BKGNDRECT

Creates a rectangle filled with the color of the background.

SS_FGNDFRAME

Creates a box with frame color equal to the foreground color.

SS_BKGNDFRAME

Creates a box with frame color equal to the background color.

SS_HALFTONERECT

Creates a rectangle filled with halftone shading.

SS_HALFTONEFRAME

Creates a box with halftone shading frame.

SS_AUTOSIZE

The static control will be sized to make sure the contents fit.

Default Colors

The following system colors are used when the system draws static controls:

SYSCLR_WINDOWFRAME

SYSCLR_WINDOWSTATICTEXT
SYSCLR_WINDOW
SYSCLR_BACKGROUND.

Some of these defaults can be replaced by using the following presentation parameters in the application resource script file or source code:

PP_BORDERCOLOR
PP_FOREGROUNDCOLOR.

Static Control Notification Messages

No notification messages are initiated by the static control window procedure.

Static Control Window Messages

This section describes the static control window procedure actions on receiving the following messages.

SM_QUERYHANDLE

SM_QUERYHANDLE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

SM_QUERYHANDLE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

SM_QUERYHANDLE Return Value - hbmHandle

hbmHandle ([HBITMAP](#))
Icon or bit-map handle of the static control.

NULLHANDLE	No icon or bit-map handle of the static control exists, or an error occurred.
Other	Icon or bit-map handle of the static control.

SM_QUERYHANDLE - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

hbmHandle ([HBITMAP](#))
Icon or bit-map handle of the static control.

NULLHANDLE	No icon or bit-map handle of the static control exists, or an error occurred.
Other	Icon or bit-map handle of the static control.

SM_QUERYHANDLE - Syntax

This message returns the icon or bit-map handle of a static control.

```
param1
    ULONG    ulReserved /* Reserved value, should be 0. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */
```

SM_QUERYHANDLE - Remarks

The static control window procedure responds to this message by setting *hbmHandle* to the handle of the icon or bit-map of the static control.

SM_QUERYHANDLE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *hbmHandle* to the default value of NULLHANDLE.

SM_QUERYHANDLE - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

SM_SETHANDLE

SM_SETHANDLE Field - hbmHandle

hbmHandle ([HBITMAP](#))
Icon or bit-map handle of a static control.

This is an icon handle when sent to a control with a style of `SS_ICON` or `SS_SYSICON`, and a bit-map handle when sent to a control with a style of `SS_BITMAP`.

SM_SETHANDLE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

SM_SETHANDLE Return Value - hbmHandle

hbmHandle ([HBITMAP](#))
Icon or bit-map handle of the static control.

<code>NULLHANDLE</code>	No icon or bit-map handle of the static control exists, or an error occurred.
Other	Icon or bit-map handle of the static control.

SM_SETHANDLE - Parameters

hbmHandle ([HBITMAP](#))
Icon or bit-map handle of a static control.

This is an icon handle when sent to a control with a style of `SS_ICON` or `SS_SYSICON`, and a bit-map handle when sent to a control with a style of `SS_BITMAP`.

ulReserved ([ULONG](#))
Reserved value, should be 0.

hbmHandle ([HBITMAP](#))
Icon or bit-map handle of the static control.

`NULLHANDLE` No icon or bit-map handle of the static control exists, or an error occurred.

Other Icon or bit-map handle of the static control.

SM_SETHANDLE - Syntax

This message sets the icon or bit-map handle of a static control.

```
param1
    HBITMAP  hbmHandle    /* Icon or bit-map handle of a static control. */

param2
    ULONG    ulReserved   /* Reserved value, should be 0. */
```

SM_SETHANDLE - Remarks

The static control window procedure responds to this message by setting the icon or bit-map handle of a static control to the value specified by *hbmHandle*, and causes the static control to be redrawn, using the new item handle.

It should only be sent to a control with a style of `SS_BITMAP`, `SS_ICON`, or `SS_SYSICON`.

SM_SETHANDLE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *hbmHandle* to the default value of `NULLHANDLE`.

SM_SETHANDLE - Topics

Select an item:
[Syntax](#)

WM_MATCHMNEMONIC (in Static Controls)

WM_MATCHMNEMONIC (in Static Controls) - Syntax

For the cause of this message, see [WM_MATCHMNEMONIC](#).

For a description of the parameters, see [WM_MATCHMNEMONIC](#).

WM_MATCHMNEMONIC (in Static Controls) - Remarks

The static control window procedure responds to this message by setting *rc* as appropriate.

WM_MATCHMNEMONIC (in Static Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_MATCHMNEMONIC (in Static Controls) - Related Messages

Related Messages

- [WM_MATCHMNEMONIC](#)
-

WM_MATCHMNEMONIC (in Static Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)

WM_QUERYCONVERTPOS (in Static Controls)

WM_QUERYCONVERTPOS (in Static Controls) - Syntax

For the cause of this message, see [WM_QUERYCONVERTPOS](#).

For a description of the parameters, see [WM_QUERYCONVERTPOS](#).

WM_QUERYCONVERTPOS (in Static Controls) - Remarks

The static control window procedure returns QCP_NOCONVERT.

WM_QUERYCONVERTPOS (in Static Controls) - Default Processing

For the default window procedure processing of this message see [WM_QUERYCONVERTPOS](#).

WM_QUERYCONVERTPOS (in Static Controls) - Related Messages

Related Messages

- [WM_QUERYCONVERTPOS](#)
-

WM_QUERYCONVERTPOS (in Static Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)

WM_QUERYWINDOWPARAMS (in Static Controls)

WM_QUERYWINDOWPARAMS (in Static Controls) - Syntax

This message occurs when an application queries the static control window procedure window parameters.

For a description of the parameters, see [WM_QUERYWINDOWPARAMS](#).

WM_QUERYWINDOWPARAMS (in Static Controls) - Remarks

The static control window procedure responds to this message by passing it to the default window procedure.

WM_QUERYWINDOWPARAMS (in Static Controls) - Default Processing

The default window procedure sets the *cchText*, *cbPresParams*, and *cbCtlData* parameters of the [WNDPARAMS](#) data structure, identified by *pwndparams*, to zero and sets *rc* to FALSE.

WM_QUERYWINDOWPARAMS (in Static Controls) - Related Messages

Related Messages

- [WM_QUERYWINDOWPARAMS](#)

WM_QUERYWINDOWPARAMS (in Static Controls) - Topics

Select an item:

[Syntax](#)

[Remarks](#)

[Default Processing](#)

WM_SETWINDOWPARAMS (in Static Controls)

WM_SETWINDOWPARAMS (in Static Controls) - Syntax

This message occurs when an application sets or changes the static control window procedure window parameters.

For a description of the parameters, see [WM_SETWINDOWPARAMS](#).

WM_SETWINDOWPARAMS (in Static Controls) - Remarks

The static control window procedure responds to this message by passing it to the default window procedure.

WM_SETWINDOWPARAMS (in Static Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_SETWINDOWPARAMS (in Static Controls) - Related Messages

Related Messages

- [WM_SETWINDOWPARAMS](#)
-

WM_SETWINDOWPARAMS (in Static Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

Title Bar Control Window Processing

This system-provided window procedure processes the actions on a title bar control (WC_TITLEBAR).

Purpose

The title bar control is the frame control that is used to display the application window title. It is also used to display the active or inactive status of the frame window.

The title bar control also implements the user interface for moving the frame window.

The standard identifier for a title bar control in a frame window is FID_TITLEBAR.

Title Bar Control Styles

There is only one title bar style, the default.

The following system colors are used when the system draws button controls:

- SYSCLR_ACTIVETITLETEXTBGND
- SYSCLR_ACTIVETITLE
- SYSCLR_ACTIVETITLETEXT
- SYSCLR_ACTIVETITLETEXTBGND
- SYSCLR_INACTIVETITLE
- SYSCLR_INACTIVETITLETEXT
- SYSCLR_INACTIVETITLETEXTBGND
- SYSCLR_TITLEBOTTOM
- SYSCLR_(IN)ACTIVETITLETEXTBGND
- SYSCLR_(IN)ACTIVETITLE.

Some of these defaults can be replaced by using the following presentation parameters in the application resource script file or source code:

- PP_FONTNAMESIZE
- PP_ACTIVECOLOR
- PP_INACTIVECOLOR
- PP_ACTIVETEXT*COLOR
- PP_INACTIVETEXT*COLOR
- PP_ACTIVETEXTFGNDCOLOR
- PP_INACTIVETEXTFGNDCOLOR
- PP_BORDERCOLOR.

Title Bar Control Notification Messages

These messages are initiated by the title bar control to notify its owner of significant events.

WM_SYSCOMMAND (in Title Bar Controls)

WM_SYSCOMMAND (in Title Bar Controls) - Syntax

For the cause of this message, see [WM_SYSCOMMAND](#).

For a description of the parameters, see [WM_SYSCOMMAND](#).

The title bar control window procedure sets *uscmd* to the title bar control identity and *ussource* to CMDSRC_OTHER.

WM_SYSCOMMAND (in Title Bar Controls) - Remarks

The title bar control window procedure generates this message when a mouse input message is received. The window procedure posts the message to the queue of the window owner.

The purpose of this message is to notify the owner window to maximize or restore depending on its current state.

WM_SYSCOMMAND (in Title Bar Controls) - Default Processing

The default window procedure takes no action on this message, other than to set *uiReserved* to 0.

WM_SYSCOMMAND (in Title Bar Controls) - Related Messages

Related Messages

- [WM_COMMAND](#)

WM_SYSCOMMAND (in Title Bar Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_TRACKFRAME (in Title Bar Controls)

WM_TRACKFRAME (in Title Bar Controls - Syntax

For the cause of this message, see [WM_TRACKFRAME](#).

For a description of the parameters, see [WM_TRACKFRAME](#).

WM_TRACKFRAME (in Title Bar Controls - Remarks

The title bar control window procedure generates this message and sends it to its owner, informing the owner that a mouse button down message has been received.

WM_TRACKFRAME (in Title Bar Controls - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_TRACKFRAME (in Title Bar Controls - Related Messages

Related Messages

- [WM_QUERYTRACKINFO](#)
-

WM_TRACKFRAME (in Title Bar Controls - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

Title Bar Control Window Messages

This section describes the title bar control window procedure actions on receiving the following messages.

TBM_QUERYHILITE

TBM_QUERYHILITE Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

TBM_QUERYHILITE Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

TBM_QUERYHILITE Return Value - rc

rc (BOOL)
Highlighting state.

TRUE	Title-bar control is highlighted
FALSE	Title-bar control is not highlighted.

TBM_QUERYHILITE - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

rc (BOOL)
Highlighting state.

TRUE	Title-bar control is highlighted
FALSE	Title-bar control is not highlighted.

TBM_QUERYHILITE - Syntax

This message returns the highlighting state of a title-bar control.

```
param1
    ULONG  ulReserved /* Reserved value, should be 0. */

param2
    ULONG  ulReserved /* Reserved value, should be 0. */
```

TBM_QUERYHILITE - Remarks

The title bar control window procedure responds to this message by returning the highlighting state of the title-bar window.

TBM_QUERYHILITE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

TBM_QUERYHILITE - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

TBM_SETHILITE

TBM_SETHILITE Field - usHighlighted

usHighlighted ([USHORT](#))
Highlighting indicator.

TRUE

FALSE	Highlight the title-bar control
	Remove highlight from the title-bar control.

TBM_SETHILITE Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

TBM_SETHILITE Return Value - rc

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

TBM_SETHILITE - Parameters

usHighlighted (USHORT)
Highlighting indicator.

TRUE	Highlight the title-bar control
FALSE	Remove highlight from the title-bar control.

ulReserved (ULONG)
Reserved value, should be 0.

rc (BOOL)
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

TBM_SETHILITE - Syntax

This message is used to highlight or unhighlight a title-bar control.

```
param1
    USHORT  usHighlighted /* Highlighting indicator. */

param2
    ULONG   ulReserved    /* Reserved value, should be 0. */
```

TBM_SETHILITE - Remarks

The title bar control window procedure responds to this message by setting the highlighting state according to *usHighlighted*. If the title bar highlighting state is changed by this message, the title bar will repaint.

TBM_SETHILITE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

TBM_SETHILITE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_QUERYCONVERTPOS (in Title Bar Controls)

WM_QUERYCONVERTPOS (in Title Bar Controls) - Syntax

For the cause of this message, see [WM_QUERYCONVERTPOS](#).

For a description of the parameters, see [WM_QUERYCONVERTPOS](#).

WM_QUERYCONVERTPOS (in Title Bar Controls) - Remarks

The title bar control window procedure returns QCP_NOCONVERT.

WM_QUERYCONVERTPOS (in Title Bar Controls) - Default Processing

For the default window procedure processing of this message see [WM_QUERYCONVERTPOS](#).

WM_QUERYCONVERTPOS (in Title Bar Controls) - Related Messages

Related Messages

- [WM_QUERYCONVERTPOS](#)
-

WM_QUERYCONVERTPOS (in Title Bar Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_QUERYWINDOWPARAMS (in Title Bars)

WM_QUERYWINDOWPARAMS (in Title Bars) - Syntax

This message occurs when an application queries the title bar control window procedure window parameters.

For a description of the parameters, see [WM_QUERYWINDOWPARAMS](#).

WM_QUERYWINDOWPARAMS (in Title Bars) - Default Processing

The title bar control window procedure queries the appropriate window parameters in accordance with *pwndparams* and sets *rc* to TRUE if the operation is successful, otherwise to FALSE.

WM_QUERYWINDOWPARAMS (in Title Bars) - Related Messages

Related Messages

- [WM_QUERYWINDOWPARAMS](#)
-

WM_QUERYWINDOWPARAMS (in Title Bars) - Topics

Select an item:

[Syntax](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_SETWINDOWPARAMS (in Title Bar Controls)

WM_SETWINDOWPARAMS (in Title Bar Controls) - Syntax

This message occurs when an application sets or changes the title bar control window procedure window parameters.

For a description of the parameters, see [WM_SETWINDOWPARAMS](#).

WM_SETWINDOWPARAMS (in Title Bar Controls) - Default Processing

The title bar control window procedure sets the appropriate window parameters in accordance with *pwndparams* and sets *rc* to TRUE if the operation is successful, otherwise to FALSE.

WM_SETWINDOWPARAMS (in Title Bar Controls) - Related Messages

Related Messages

- [WM_SETWINDOWPARAMS](#)

WM_SETWINDOWPARAMS (in Title Bar Controls) - Topics

Select an item:

[Syntax](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

Value Set Control Window Processing

This system-provided window procedure processes the actions on a value set control (WC_VALUESET).

Purpose

Like radio buttons, a value set control (WC_VALUESET window class) is a visual component whose specific purpose is to allow a user to select one choice from a group of mutually exclusive choices. However, unlike radio buttons, a value set can use graphical images (bit maps or icons), as well as colors, text, and numbers, to represent the items that a user can select.

Even though text is supported, a value set's primary purpose is to display choices as graphical images. By using graphical images in a value set, you can preserve space on the display screen. You can also allow the user to see exactly what is being selected instead of having to rely on descriptions of the choices. This allows a user to make a selection faster than if the user had to read a description of each choice. For example, if you want to allow a user to choose from a variety of patterns, you can present those patterns as value set choices instead of having to provide a list of radio buttons with description of each pattern.

If long strings of data are to be displayed as choices, radio buttons should be used. However, for small sets of numeric or textual data information, either a value set or radio buttons can be used.

The value set is designed to be customizable to meet varying application requirements, while providing an easy-to-use user interface component that can be used to develop products that conform to the Common User Access (CUA) user interface guidelines. The application can specify different types of items, sizes, and orientations for its value sets, but the underlying function of the control remains the same. For a complete description of CUA value sets, refer to the *SAA CUA Guide to User Interface Design* and the *SAA CUA Advanced Interface Design Reference*.

Value Set Control Styles

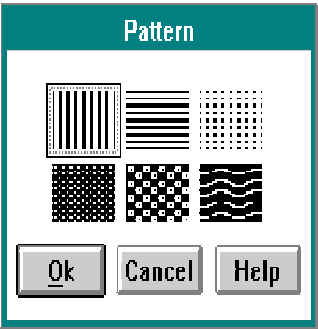
Value set control window styles are set when a value set window is created.

- Set one of the following styles when creating a value set control window. You can override these styles by specifying VIA_BITMAP, VIA_ICON, VIA_TEXT, VIA_RGB, or VIA_COLORINDEX attributes for individual value set items.

VS_BITMAP

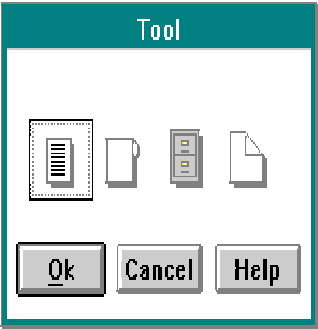
The attribute for each value set item is set to the VIA_BITMAP value set item attribute, which means the value set treats each item as a bit map unless otherwise specified. This is the default. The following picture provides an example of a value set with bit maps.

VS_ICON



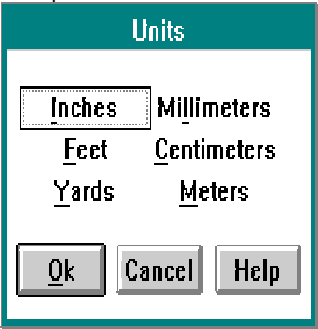
The attribute for each value set item is set to the VIA_ICON value set item attribute, which means the value set treats each item as an icon unless otherwise specified. The following picture provides an example of a value set with icons.

VS_TEXT



The attribute for each value set item is set to the VIA_TEXT value set item attribute, which means the value set treats each item as a text string unless otherwise specified. The following picture provides an example of a value set with text strings.

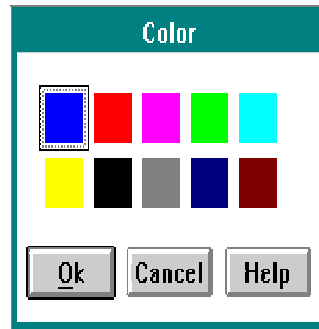
VS_RGB



The attribute for each value set item is set to the VIA_RGB value set item attribute, which means the value set treats each item as a RGB color value unless otherwise specified. This style is most often used when you need to create new colors. The picture following the definition of VS_COLORINDEX provides an example of a value set with colors.

VS_COLORINDEX

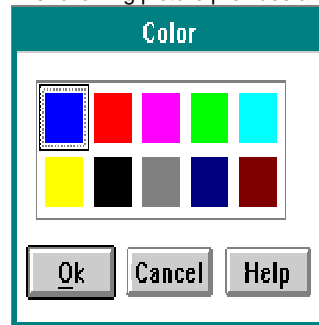
The attribute for each value set item is set to the VIA_COLORINDEX value set item attribute, which means the value set treats each item as an index into the logical color table unless otherwise specified. This style is most often used when the colors currently available are adequate. The following picture provides an example of a value set with colors.



- Specify one or more of the following optional window styles, if desired, by using an OR operator (|) to combine them with the style specified from the preceding list:

VS_BORDER

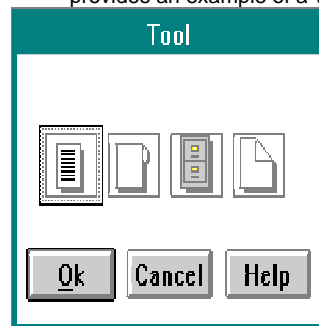
The value set draws a thin border around itself to delineate the control. The following picture provides an example of a value set with a border.



VS_ITEMBORDER

The value set draws a thin border around each item to delineate it from other items.

Note: The VS_ITEMBORDER style is useful for items that are hard to see, such as faint colors or patterns. The following picture provides an example of a value set with item borders.



VS_RIGHTTOLEFT

The value set interprets column orientation as right-to-left, instead of the default left-to-right arrangement. This means columns are numbered from right-to-left with the rightmost column being 1 and counting up as you move left. Home is the rightmost column and end is the leftmost column.

There is no visible difference between a value set ordered left-to-right and a value set ordered right-to-left. Therefore, if your application uses multiple value sets, the ordering of the items should be consistent in each value set to avoid confusing the user.

Note: The VS_RIGHTTOLEFT style is used on creation of the control. Changing this style after creation causes unexpected results.

VS_SCALEBITMAPS

The value set automatically scales bit maps to the size of the cell. If this style is not used, each bit map is centered in its cell. Also, if the cell is smaller than the bit map, the bit map is clipped to the size of the cell.

VS_OWNERDRAW

The application is notified whenever the background of the value set window is to be painted.

Value Set Control Data

For information on value set control data, see the following:

- [VSCDATA](#)
 - [VSDRAGINFO](#)
 - [VSDRAGINIT](#)
 - [VSTEXT](#).
-

Value Set Control Notification Messages

These messages are initiated by the value set control window to notify its owner of significant events.

WM_CONTROL (in Value Set Controls)

WM_CONTROL (in Value Set Controls) Field - id

id ([USHORT](#))
Value set control identity.

WM_CONTROL (in Value Set Controls) Field - notifycode

notifycode ([USHORT](#))
Notify code.

The value set control uses these notification codes:

VN_DRAGLEAVE
The value set receives a [DM_DRAGLEAVE](#) message.

VN_DRAGOVER
The value set receives a [DM_DRAGOVER](#) message.

VN_DROP
The value set receives a [DM_DROP](#) message. The VN_DROP notification code is sent only when an item is dropped on an item that has the VIA_DROPONABLE attribute.

VN_DROPHELP
The value set receives a [DM_DROPHELP](#) message.

VN_ENTER	The user presses the Enter key while the value set window has the focus or double-clicks the select button while the pointer is over an item in the value set.
VN_HELP	The value set receives a WM_HELP message.
VN_INITDRAG	The drag button was pressed and the pointer was moved while the pointer was over the value set control. The VN_INITDRAG notification code is sent only for items that have the VIA_DRAGGABLE attribute.
VN_KILLFOCUS	The value set is losing the focus.
VN_SELECT	An item in the value set has been selected and is given selected-state emphasis.
VN_SETFOCUS	The value set receives the focus.

WM_CONTROL (in Value Set Controls) Field - notifyinfo

notifyinfo ([ULONG](#))

Control-specific information.

When the value of the *notifycode* parameter is VN_DRAGOVER, VN_DRAGLEAVE, VN_DROP, or VN_DROPHELP, this parameter is a pointer to a [VSDRAGINFO](#) structure.

When the value of the *notifycode* parameter is VN_INITDRAG, this parameter is a pointer to a [VSDRAGINIT](#) structure.

When the value of the *notifycode* parameter is VN_ENTER, VN_HELP, or VN_SELECT, this parameter contains the row and column of the selection cursor. The low-order word contains the row index, and the high-order word contains the column index.

Otherwise, this parameter is the window handle ([HWND](#)) of the value set control.

WM_CONTROL (in Value Set Controls) Return Value - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_CONTROL (in Value Set Controls) - Parameters

id ([USHORT](#))

Value set control identity.

notifycode ([USHORT](#))

Notify code.

The value set control uses these notification codes:

VN_DRAGLEAVE	The value set receives a DM_DRAGLEAVE message.
VN_DRAGOVER	The value set receives a DM_DRAGOVER message.
VN_DROP	The value set receives a DM_DROP message. The VN_DROP notification code is sent only when an item is dropped on an item that has the VIA_DROPONABLE attribute.
VN_DROPHELP	The value set receives a DM_DROPHELP message.
VN_ENTER	The user presses the Enter key while the value set window has the focus or double-clicks the select button while the pointer is over an item in the value set.
VN_HELP	The value set receives a WM_HELP message.
VN_INITDRAG	The drag button was pressed and the pointer was moved while the pointer was over the value set control. The VN_INITDRAG notification code is sent only for items that have the VIA_DRAGGABLE attribute.
VN_KILLFOCUS	The value set is losing the focus.
VN_SELECT	An item in the value set has been selected and is given selected-state emphasis.
VN_SETFOCUS	The value set receives the focus.

notifyinfo (ULONG)

Control-specific information.

When the value of the *notifycode* parameter is VN_DRAGOVER, VN_DRAGLEAVE, VN_DROP, or VN_DROPHELP, this parameter is a pointer to a [VSDRAGINFO](#) structure.

When the value of the *notifycode* parameter is VN_INITDRAG, this parameter is a pointer to a [VSDRAGINIT](#) structure.

When the value of the *notifycode* parameter is VN_ENTER, VN_HELP, or VN_SELECT, this parameter contains the row and column of the selection cursor. The low-order word contains the row index, and the high-order word contains the column index.

Otherwise, this parameter is the window handle ([HWND](#)) of the value set control.

ulReserved (ULONG)

Reserved value, should be 0.

WM_CONTROL (in Value Set Controls) - Syntax

For the cause of this message, see [WM_CONTROL](#).

```
param1
    USHORT id           /* Value set control identity. */
    USHORT notifycode    /* Notify code. */

param2
    ULONG notifyinfo     /* Control-specific information. */
```

WM_CONTROL (in Value Set Controls) - Remarks

The value set control window procedure generates this message and sends it to its owner, informing the owner of this event.

WM_CONTROL (in Value Set Controls) - Default Processing

For a description of the default processing, see [WM_CONTROL](#).

WM_CONTROL (in Value Set Controls) - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

WM_CONTROLPOINTER (in Value Set Controls)

WM_CONTROLPOINTER (in Value Set Controls) - Syntax

For the cause of this message, see [WM_CONTROLPOINTER](#).

For a description of the parameters, see [WM_CONTROLPOINTER](#).

WM_CONTROLPOINTER (in Value Set Controls) - Remarks

For the appropriate remarks, see [WM_CONTROLPOINTER](#).

WM_CONTROLPOINTER (in Value Set Controls) - Default Processing

For the default processing, see [WM_CONTROLPOINTER](#).

WM_CONTROLPOINTER (in Value Set Controls) - Topics

Select an item:

[Syntax](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

WM_DRAWITEM (in Value Set Controls)

WM_DRAWITEM (in Value Set Controls) Field - id

id ([USHORT](#))

Window identifier.

The window identifier of the value set control sending this notification message.

WM_DRAWITEM (in Value Set Controls) Field - powneritem

powneritem ([OWNERITEM](#))

Pointer to an [OWNERITEM](#) data structure.

The following list defines the [OWNERITEM](#) data structure fields that apply to the value set control. See [OWNERITEM](#) for the default field values.

hwnd ([HWND](#))

Value set window handle.

hps ([HPS](#))

Presentation-space handle.

fsState ([ULONG](#))

Value set window style flags. See [Value Set Control Styles](#) for descriptions of these style flags.

fsAttribute ([ULONG](#))

Item attribute flags for the indexed item. See [VM_SETITEMATTR](#) for descriptions of these attribute flags.

fsStateOld ([ULONG](#))

Reserved.

fsAttributeOld ([ULONG](#))

Reserved.

rcItem ([RECTL](#))

Item rectangle to be drawn in window coordinates.

idlItem ([LONG](#))

Identity of component to be drawn.

VDA_BACKGROUND

Specifies that a part of the value set background is to be drawn.

VDA_SURROUNDING

Specifies that a part of the area surrounding the value set is to be drawn.

VDA_ITEMBACKGROUND

Specifies that the background of an item is to be drawn.

VDA_ITEM

Specifies that an entire item is to be drawn.

hlItem ([ULONG](#))

If the value of the **identity** parameter is VDA_ITEMBACKGROUND or VDA_ITEM, this is the current row and column index of the item to be drawn. The low-order word contains the row index, and the high-order word contains the column index. Otherwise, this is reserved.

WM_DRAWITEM (in Value Set Controls) Return Value - rc

rc ([BOOL](#))

Item-drawn indicator.

TRUE

The owner draws the component.

FALSE

If the owner does not draw the component, the owner returns this value and the value set control draws the component.

WM_DRAWITEM (in Value Set Controls) - Parameters

id ([USHORT](#))

Window identifier.

The window identifier of the value set control sending this notification message.

powneritem ([POWNERITEM](#))

Pointer to an [OWNERITEM](#) data structure.

The following list defines the [OWNERITEM](#) data structure fields that apply to the value set control. See [OWNERITEM](#) for the default field values.

hwnd ([HWND](#))

Value set window handle.

hps ([HPS](#))

Presentation-space handle.

fsState ([ULONG](#))

Value set window style flags. See [Value Set Control Styles](#) for descriptions of these style flags.

fsAttribute ([ULONG](#))

Item attribute flags for the indexed item. See [VM_SETITEMATTR](#) for descriptions of these attribute flags.

fsStateOld ([ULONG](#))

Reserved.

<i>fsAttributeOld</i> (ULONG)	Reserved.	
<i>rcItem</i> (RECTL)	Item rectangle to be drawn in window coordinates.	
<i>idItem</i> (LONG)	Identity of component to be drawn.	
	VDA_BACKGROUND	Specifies that a part of the value set background is to be drawn.
	VDA_SURROUNDING	Specifies that a part of the area surrounding the value set is to be drawn.
	VDA_ITEMBACKGROUND	Specifies that the background of an item is to be drawn.
	VDA_ITEM	Specifies that an entire item is to be drawn.
<i>hItem</i> (ULONG)	If the value of the identity parameter is VDA_ITEMBACKGROUND or VDA_ITEM, this is the current row and column index of the item to be drawn. The low-order word contains the row index, and the high-order word contains the column index. Otherwise, this is reserved.	
rc (BOOL)	Item-drawn indicator.	
	TRUE	The owner draws the component.
	FALSE	If the owner does not draw the component, the owner returns this value and the value set control draws the component.

WM_DRAWITEM (in Value Set Controls) - Syntax

This notification message is sent to the owner of a value set control each time an item that has the VIA_OWNERDRAW attribute is to be drawn, or when the background of a value set window that has the VS_OWNERDRAW style bit is to be drawn.

```
param1
    USHORT        id            /* Window identifier. */

param2
    POWNERITEM    powneritem    /* Pointer to an OWNERITEM data structure. */
```

WM_DRAWITEM (in Value Set Controls) - Remarks

The value set control draws only items that are represented in one of the formats described: text, color, bit maps, or icons.

If an application uses value set controls that contain items that are not represented by the supported formats or requires that the emphasized attribute of an item is to be drawn in a special manner, the application must specify those items as VIA_OWNERDRAW and those items must be drawn by the owner.

Through this message, the application can provide a custom value set background (the area between the items) and customize the area surrounding the value set (the area on the top and right sides of the value set that is left over when the value set calculates its size). The

application can specify how either or both of these areas are drawn and is given the opportunity to do so.

The value set control window procedure generates this message and sends it to its owner, informing the owner that something is to be drawn. The owner is given the opportunity to draw and to indicate whether the value set control should continue with the normal drawing of that component.

WM_DRAWITEM (in Value Set Controls) - Default Processing

For a description of the default processing, see [WM_DRAWITEM](#).

WM_DRAWITEM (in Value Set Controls) - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

Value Set Control Window Messages

This section describes the value set control window procedure actions on receiving the following messages.

VM_QUERYITEM

VM_QUERYITEM Field - usRow

usRow ([USHORT](#))

Row index.

Row index of the item to be queried. Rows have a value from 1 to the value of the *usRowCount* field. This value, which is the total number of rows in the value set, is specified in the [VSCDATA](#) data structure when the value set control is created.

VM_QUERYITEM Field - usColumn

usColumn (USHORT)
Column index.

Column index of the item to be queried. Columns have a value from 1 to the value of the *usColumnCount* field. This value, which is the total number of columns in the value set, is specified in the **VSCDATA** data structure when the value set control is created.

VM_QUERYITEM Field - pvsText

pvsText (PVSTEXT)
Pointer to a **VSTEXT** data structure or NULL.

If the attribute of the item to query is VIA_TEXT, the value of the *param2* parameter is the same as the value of the *pvsText* field. For all other attributes, the *param2* parameter is reserved and should be set to a NULL value.

VM_QUERYITEM Return Value - ullItemId

ullItemId (ULONG)
Item information.

This value depends on the VIA_* attribute specified for the value set item.

- If the VIA_TEXT attribute is set, the following is returned:
usTextLen (USHORT)
Number of bytes copied to the buffer. This is the length of the text string, excluding the null termination character.
- If the VIA_BITMAP attribute is set, the following is returned:
hbmItem (HBITMAP)
Handle of the bit map associated with the item indexed by the *param1* parameter. If the item is empty, a NULL value is returned.
- If the VIA_ICON attribute is set, the following is returned:
hptItem (HPOINTER)
Handle of the icon associated with the item indexed by the *param1* parameter. If the item is empty, a NULL value is returned.
- If the VIA_RGB attribute is set, the following is returned:
rgbItem (ULONG)
Color value associated with the item indexed by the *param1* parameter. If the item is empty, a NULL value is returned. Each color value is a 4-byte integer with a value of:
$$(R * 65536) + (G * 256) + B$$

where:

R	Red intensity value
G	Green intensity value
B	Blue intensity value.
- If the VIA_COLORINDEX attribute is set, the following is returned:
ulColorIndex (ULONG)
Index of the color associated with the item indexed by the *param1* parameter.

The following is returned for any of the items to indicate an error condition:

VSERR_INVALID_PARAMETERS

An error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_PARAMETER_OUT_OF_RANGE.

VM_QUERYITEM - Parameters

usRow ([USHORT](#))

Row index.

Row index of the item to be queried. Rows have a value from 1 to the value of the *usRowCount* field. This value, which is the total number of rows in the value set, is specified in the [VSCDATA](#) data structure when the value set control is created.

usColumn ([USHORT](#))

Column index.

Column index of the item to be queried. Columns have a value from 1 to the value of the *usColumnCount* field. This value, which is the total number of columns in the value set, is specified in the [VSCDATA](#) data structure when the value set control is created.

pvsText ([PVSTEXT](#))

Pointer to a [VSTEXT](#) data structure or NULL.

If the attribute of the item to query is VIA_TEXT, the value of the *param2* parameter is the same as the value of the *pvsText* field. For all other attributes, the *param2* parameter is reserved and should be set to a NULL value.

ulItemId ([ULONG](#))

Item information.

This value depends on the VIA_* attribute specified for the value set item.

- If the VIA_TEXT attribute is set, the following is returned:

usTextLen ([USHORT](#))

Number of bytes copied to the buffer. This is the length of the text string, excluding the null termination character.

- If the VIA_BITMAP attribute is set, the following is returned:

hbmItem ([HBITMAP](#))

Handle of the bit map associated with the item indexed by the *param1* parameter. If the item is empty, a NULL value is returned.

- If the VIA_ICON attribute is set, the following is returned:

hptItem ([HPOINTER](#))

Handle of the icon associated with the item indexed by the *param1* parameter. If the item is empty, a NULL value is returned.

- If the VIA_RGB attribute is set, the following is returned:

rgbItem ([ULONG](#))

Color value associated with the item indexed by the *param1* parameter. If the item is empty, a NULL value is returned. Each color value is a 4-byte integer with a value of:

$(R * 65536) + (G * 256) + B$

where:

R	Red intensity value
G	Green intensity value

B Blue intensity value.

- If the `VIA_COLORINDEX` attribute is set, the following is returned:

uiColorIndex ([ULONG](#))
Index of the color associated with the item indexed by the *param1* parameter.

The following is returned for any of the items to indicate an error condition:

`VSERR_INVALID_PARAMETERS`

An error occurred. The [WinGetLastError](#) function may return the following errors:

- `PMERR_INVALID_PARAMETERS`
- `PMERR_PARAMETER_OUT_OF_RANGE`.

VM_QUERYITEM - Syntax

This message queries the contents of the item indicated by the values of the *usRow* and *usColumn* fields. The information returned is interpreted based on the attribute of the item.

```
param1
    USHORT    usRow        /* Row index. */
    USHORT    usColumn     /* Column index. */

param2
    PVSTEXT   pvsText      /* Pointer to a VSTEXT data structure or NULL. */
```

VM_QUERYITEM - Remarks

The application uses this message to query the contents of an individual value set item. When querying a text item, the application must provide a buffer for returning the text information. By specifying 0 as the value of the *usBufLen* field and then getting the value returned in the *usTextLen* parameter, an application can determine how large this buffer must be. The value returned is the length of the text string, excluding the null termination character.

VM_QUERYITEM - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

VM_QUERYITEM - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)

VM_QUERYITEMATTR

VM_QUERYITEMATTR Field - usRow

usRow ([USHORT](#))
Row index.

Row index of the item for which the attribute or attributes are queried. Rows have a value from 1 to the value of the *usRowCount* field. This value, which is the total number of rows in the value set, is specified in the [VSCDATA](#) data structure when the value set control is created.

VM_QUERYITEMATTR Field - usColumn

usColumn ([USHORT](#))
Column index.

Column index of the item for which the attribute or attributes are queried. Columns have a value from 1 to the value of the *usColumnCount* field. This value, which is the total number of columns in the value set, is specified in the [VSCDATA](#) data structure when the value set control is created.

VM_QUERYITEMATTR Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

VM_QUERYITEMATTR Return Value - usItemAttr

usItemAttr ([USHORT](#))
Item information.

This value depends on the *VIA_** attribute or attributes specified for the value set item.

- One of the following attributes can be set:

- | | |
|----------------|--|
| VIA_BITMAP | If this attribute is set, the item is a bit map. This is the default. |
| VIA_COLORINDEX | If this attribute is set, the item is an index into the logical color table. |
| VIA_ICON | If this attribute is set, the item is an icon. |
| VIA_RGB | If this attribute is set, the item is a color entry. |
| VIA_TEXT | If this attribute is set, the item is a text string. |
- In addition, one or more of the following attributes can be set:

VIA_DISABLED	<p>If this attribute is set, the item cannot be selected and is displayed with unavailable-state emphasis, if possible. Unavailable text items are always displayed with unavailable-state emphasis, according to CUA guidelines; for items displayed as color, bit maps, and icons, it is the application's responsibility to determine the best way to show that these items are unavailable, if possible.</p> <p>The selection cursor can be moved to an unavailable item by using either the keyboard navigation keys or a pointing device. This allows a user to press the F1 key to find out why that item cannot be selected.</p>
VIA_DRAGGABLE	If this attribute is set, the item can be the source of a direct manipulation action.
VIA_DROPONABLE	If this attribute is set, the item can be the target of a direct manipulation action.
VIA_OWNERDRAW	If this attribute is set, a paint notification message is sent whenever this item needs painting.
 - The following is returned if an error occurs:

VMERR_INVALID_PARAMETERS	The WinGetLastError function may return the following errors:
-	PMERR_INVALID_PARAMETERS
-	PMERR_PARAMETER_OUT_OF_RANGE.

VM_QUERYITEMATTR - Parameters

usRow ([USHORT](#))

Row index.

Row index of the item for which the attribute or attributes are queried. Rows have a value from 1 to the value of the *usRowCount* field. This value, which is the total number of rows in the value set, is specified in the [VSCDATA](#) data structure when the value set control is created.

usColumn ([USHORT](#))

Column index.

Column index of the item for which the attribute or attributes are queried. Columns have a value from 1 to the value of the *usColumnCount* field. This value, which is the total number of columns in the value set, is specified in the [VSCDATA](#) data structure when the value set control is created.

ulReserved ([ULONG](#))

Reserved value, should be 0.

usItemAttr ([USHORT](#))

Item information.

This value depends on the `VIA_*` attribute or attributes specified for the value set item.

- One of the following attributes can be set:

<code>VIA_BITMAP</code>	If this attribute is set, the item is a bit map. This is the default.
<code>VIA_COLORINDEX</code>	If this attribute is set, the item is an index into the logical color table.
<code>VIA_ICON</code>	If this attribute is set, the item is an icon.
<code>VIA_RGB</code>	If this attribute is set, the item is a color entry.
<code>VIA_TEXT</code>	If this attribute is set, the item is a text string.

- In addition, one or more of the following attributes can be set:

<code>VIA_DISABLED</code>	<p>If this attribute is set, the item cannot be selected and is displayed with unavailable-state emphasis, if possible. Unavailable text items are always displayed with unavailable-state emphasis, according to CUA guidelines; for items displayed as color, bit maps, and icons, it is the application's responsibility to determine the best way to show that these items are unavailable, if possible.</p> <p>The selection cursor can be moved to an unavailable item by using either the keyboard navigation keys or a pointing device. This allows a user to press the F1 key to find out why that item cannot be selected.</p>
<code>VIA_DRAGGABLE</code>	If this attribute is set, the item can be the source of a direct manipulation action.
<code>VIA_DROPONABLE</code>	If this attribute is set, the item can be the target of a direct manipulation action.
<code>VIA_OWNERDRAW</code>	If this attribute is set, a paint notification message is sent whenever this item needs painting.

- The following is returned if an error occurs:

`VMERR_INVALID_PARAMETERS`

The [WinGetLastError](#) function may return the following errors:

- `PMERR_INVALID_PARAMETERS`
- `PMERR_PARAMETER_OUT_OF_RANGE`.

VM_QUERYITEMATTR - Syntax

This message queries the attribute or attributes of the item indicated by the values of the *usRow* and *usColumn* fields.

```
param1
    USHORT  usRow      /* Row index. */
    USHORT  usColumn   /* Column index. */

param2
    ULONG   ulReserved /* Reserved value, should be 0. */
```

VM_QUERYITEMATTR - Remarks

The application uses this message to query the specific attribute or attributes of a value set item.

VM_QUERYITEMATTR - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

VM_QUERYITEMATTR - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

VM_QUERYMETRICS

VM_QUERYMETRICS Field - fMetric

fMetric ([USHORT](#))

Control metric.

Control metric to be queried with this message. This can be either of the following:

VMA_ITEMSIZE

If this message attribute is set, the width and height of each item (in pixels) are returned in the *usItemWidth* and *usItemHeight* parameters, respectively.

VMA_ITEMSPACING

If this message attribute is set, the horizontal and vertical spacing between items (in pixels) is returned in the *usHorzItemSpacing* parameter and in the *usVertItemSpacing* parameter, respectively.

VM_QUERYMETRICS Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

VM_QUERYMETRICS Return Value - ulMetric

ulMetric (ULONG)

Metric value queried for.

VSERR_INVALID_PARAMETERS

An error occurred. The [WinGetLastError](#) function may return the following error:

PMERR_INVALID_PARAMETERS.

>= 0

This value depends on the VMA_* attribute set in the *param1* parameter.

- If the VMA_ITEMSIZE attribute is set, the following is returned:

usItemWidth (USHORT)

Width of one value set item, in pixels.

usItemHeight (USHORT)

Height of one value set item, in pixels.

- If the VMA_ITEMSPACING attribute is set, the following is returned:

usHorzItemSpacing (USHORT)

Amount of horizontal space allocated between each value set item, in pixels. This number does not include the space needed for selected-state and target emphasis, and for the selection cursor, because the emphasis and cursor space is automatically allocated by the value set control. The default space amount is 0.

usVertItemSpacing (USHORT)

Amount of vertical space allocated between each value set item, in pixels. This number does not include the space needed for selected-state and target emphasis, and for the selection cursor, because the emphasis and cursor space is automatically allocated by the value set control. The default space amount is 0.

VM_QUERYMETRICS - Parameters

fMetric (USHORT)

Control metric.

Control metric to be queried with this message. This can be either of the following:

VMA_ITEMSIZE

If this message attribute is set, the width and height of each item (in pixels) are returned in the *usItemWidth* and *usItemHeight* parameters, respectively.

VMA_ITEMSPACING

If this message attribute is set, the horizontal and vertical spacing between items (in pixels) is returned in the *usHorzItemSpacing* parameter and in the *usVertItemSpacing* parameter, respectively.

ulReserved (ULONG)

Reserved value, should be 0.

ulMetric (ULONG)

Metric value queried for.

VSERR_INVALID_PARAMETERS

An error occurred. The [WinGetLastError](#) function may return the following error:

PMERR_INVALID_PARAMETERS.

>= 0

This value depends on the VMA_* attribute set in the *param1* parameter.

- If the VMA_ITEMSIZE attribute is set, the following is returned:

usItemWidth ([USHORT](#))

Width of one value set item, in pixels.

usItemHeight ([USHORT](#))

Height of one value set item, in pixels.

- If the VMA_ITEMSPACING attribute is set, the following is returned:

usHorzItemSpacing ([USHORT](#))

Amount of horizontal space allocated between each value set item, in pixels. This number does not include the space needed for selected-state and target emphasis, and for the selection cursor, because the emphasis and cursor space is automatically allocated by the value set control. The default space amount is 0.

usVertItemSpacing ([USHORT](#))

Amount of vertical space allocated between each value set item, in pixels. This number does not include the space needed for selected-state and target emphasis, and for the selection cursor, because the emphasis and cursor space is automatically allocated by the value set control. The default space amount is 0.

VM_QUERYMETRICS - Syntax

This message queries for the current size of each value set item or for the spacing between items. The value returned is either the width and height of one item, or the spacing between items.

```
param1
    USHORT  fMetric      /* Control metric. */

param2
    ULONG    ulReserved  /* Reserved value, should be 0. */
```

VM_QUERYMETRICS - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

VM_QUERYMETRICS - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Default Processing](#)

[Glossary](#)

VM_QUERYSELECTEDITEM

VM_QUERYSELECTEDITEM Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

VM_QUERYSELECTEDITEM Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

VM_QUERYSELECTEDITEM Field - usRow

usRow ([USHORT](#))
Row index.

Row index of the currently selected value set item. Rows have a value from 1 to the value of the *usRowCount* field. This value, which is the total number of rows in the value set, is specified in the [VSCDATA](#) data structure when the value set control is created.

VM_QUERYSELECTEDITEM Field - usColumn

usColumn ([USHORT](#))
Column index.

Column index of the currently selected value set item. Columns have a value from 1 to the value of the *usColumnCount* field. This value, which is the total number of columns in the value set, is specified in the [VSCDATA](#) data structure when the value set control is created.

VM_QUERYSELECTEDITEM - Parameters

ulReserved ([ULONG](#))

Reserved value, should be 0.

ulReserved ([ULONG](#))

Reserved value, should be 0.

usRow ([USHORT](#))

Row index.

Row index of the currently selected value set item. Rows have a value from 1 to the value of the *usRowCount* field. This value, which is the total number of rows in the value set, is specified in the [VSCDATA](#) data structure when the value set control is created.

usColumn ([USHORT](#))

Column index.

Column index of the currently selected value set item. Columns have a value from 1 to the value of the *usColumnCount* field. This value, which is the total number of columns in the value set, is specified in the [VSCDATA](#) data structure when the value set control is created.

VM_QUERYSELECTEDITEM - Syntax

This message queries for the currently selected value set item indicated by the values of the *usRow* and *usColumn* fields.

```
param1
    ULONG    ulReserved /* Reserved value, should be 0. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */

returns
    USHORT   usRow       /* Row index. */
    USHORT   usColumn    /* Column index. */
```

VM_QUERYSELECTEDITEM - Remarks

The application uses this message to query the index of the currently selected value set item. If 0 is returned, no item is selected.

VM_QUERYSELECTEDITEM - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return 0.

VM_QUERYSELECTEDITEM - Topics

Select an item:

[Syntax](#)

[Parameters](#)

VM_SELECTITEM

VM_SELECTITEM Field - usRow

usRow ([USHORT](#))
Row index.

Row index of the value set item to select. Rows have a value from 1 to the value of the *usRowCount* field. This value, which is the total number of rows in the value set, is specified in the [VSCDATA](#) data structure when the value set control is created.

VM_SELECTITEM Field - usColumn

usColumn ([USHORT](#))
Column index.

Column index of the value set item to select. Columns have a value from 1 to the value of the *usColumnCount* field. This value, which is the total number of columns in the value set, is specified in the [VSCDATA](#) data structure when the value set control is created.

VM_SELECTITEM Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

VM_SELECTITEM Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE
Item was successfully selected.

FALSE

An error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_PARAMETER_OUT_OF_RANGE.

VM_SELECTITEM - Parameters

usRow ([USHORT](#))

Row index.

Row index of the value set item to select. Rows have a value from 1 to the value of the *usRowCount* field. This value, which is the total number of rows in the value set, is specified in the [VSCDATA](#) data structure when the value set control is created.

usColumn ([USHORT](#))

Column index.

Column index of the value set item to select. Columns have a value from 1 to the value of the *usColumnCount* field. This value, which is the total number of columns in the value set, is specified in the [VSCDATA](#) data structure when the value set control is created.

ulReserved ([ULONG](#))

Reserved value, should be 0.

rc ([BOOL](#))

Success indicator.

TRUE

Item was successfully selected.

FALSE

An error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_PARAMETER_OUT_OF_RANGE.

VM_SELECTITEM - Syntax

This message selects the value set item indicated by the values of the *usRow* and *usColumn* parameters. When a new item is selected, the previously selected item is deselected.

```
param1
    USHORT  usRow          /* Row index. */
    USHORT  usColumn       /* Column index. */

param2
    ULONG    ulReserved    /* Reserved value, should be 0. */
```

VM_SELECTITEM - Remarks

The application uses this message to select the specified value set item.

VM_SELECTITEM - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

VM_SELECTITEM - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

VM_SETITEM

VM_SETITEM Field - usRow

usRow ([USHORT](#))
Row index.

Row index of the value set item for which information is being specified. Rows have a value from 1 to the value of the *usRowCount* field. This value, which is the total number of rows in the value set, is specified in the [VSCDATA](#) data structure when the value set control is created.

VM_SETITEM Field - usColumn

usColumn ([USHORT](#))
Column index.

Column index of the value set item for which information is being specified. Columns have a value from 1 to the value of the *usColumnCount* field. This value, which is the total number of columns in the value set, is specified in the [VSCDATA](#) data structure when the value set control is created.

VM_SETITEM Field - ulltemId

ulltemId (ULONG)
Item information.

This value depends on the VIA_* attribute set for the item.

- If the VIA_TEXT attribute is specified, the *ulltemId* field is as follows:

pszItem (PSZ)
Pointer to a null terminated string containing the text to be placed in the item. If NULL is passed in, the item is blank.
- If the VIA_BITMAP attribute is specified, the *ulltemId* field is as follows:

hbmItem (HBITMAP)
Handle to a bit map that is to be drawn in the item indicated by the *param1* parameter. If NULLHANDLE is passed in, the item will be blank.
- If the VIA_ICON attribute is specified, the *ulltemId* field is as follows:

hptItem (HPOINTER)
Handle to the icon that is to be drawn in the item indicated by the *param1* parameter. If NULLHANDLE is passed in, the item is blank.
- If the VIA_RGB attribute is specified, the *ulltemId* field is as follows:

rgbItem (ULONG)
Color value to be drawn in the item indicated by the *param1* parameter. If an invalid value is passed in (a value greater than 0x00FFFFFF), the item is blank. Each color value is a 4-byte integer with a value of:

$$(R * 65536) + (G * 256) + B$$

where:

R	Red intensity value
G	Green intensity value
B	Blue intensity value.
- If the VIA_COLORINDEX attribute is specified, the *ulltemId* field is as follows:

ulColorIndex (ULONG)
Index of the color in the logical color table to be drawn in the item indicated by the *param1* parameter.

VM_SETITEM Return Value - rc

rc (BOOL)
Success indicator.

TRUE
Item was successfully set.

FALSE
An error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_PARAMETER_OUT_OF_RANGE.

VM_SETITEM - Parameters

usRow (**USHORT**)
Row index.

Row index of the value set item for which information is being specified. Rows have a value from 1 to the value of the *usRowCount* field. This value, which is the total number of rows in the value set, is specified in the **VSCDATA** data structure when the value set control is created.

usColumn (**USHORT**)
Column index.

Column index of the value set item for which information is being specified. Columns have a value from 1 to the value of the *usColumnCount* field. This value, which is the total number of columns in the value set, is specified in the **VSCDATA** data structure when the value set control is created.

ulItemId (**ULONG**)
Item information.

This value depends on the VIA_* attribute set for the item.

- If the VIA_TEXT attribute is specified, the *ulItemId* field is as follows:

pszItem (**PSZ**)

Pointer to a null terminated string containing the text to be placed in the item. If NULL is passed in, the item is blank.

- If the VIA_BITMAP attribute is specified, the *ulItemId* field is as follows:

hbmItem (**HBITMAP**)

Handle to a bit map that is to be drawn in the item indicated by the *param1* parameter. If NULLHANDLE is passed in, the item will be blank.

- If the VIA_ICON attribute is specified, the *ulItemId* field is as follows:

hptItem (**HPOINTER**)

Handle to the icon that is to be drawn in the item indicated by the *param1* parameter. If NULLHANDLE is passed in, the item is blank.

- If the VIA_RGB attribute is specified, the *ulItemId* field is as follows:

rgbItem (**ULONG**)

Color value to be drawn in the item indicated by the *param1* parameter. If an invalid value is passed in (a value greater than 0x00FFFFFF), the item is blank. Each color value is a 4-byte integer with a value of:

$$(R * 65536) + (G * 256) + B$$

where:

R	Red intensity value
G	Green intensity value
B	Blue intensity value.

- If the VIA_COLORINDEX attribute is specified, the *ulItemId* field is as follows:

uiColorIndex (**ULONG**)

Index of the color in the logical color table to be drawn in the item indicated by the *param1* parameter.

rc (**BOOL**)
Success indicator.

TRUE
Item was successfully set.

FALSE

An error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_PARAMETER_OUT_OF_RANGE.

VM_SETITEM - Syntax

This message specifies the type of information that will be contained by a value set item. This item is indicated by the values of the *usRow* and *usColumn* fields. Each value set item can contain a different type of information. The value set interprets the information set for the item based on the attribute of the item. Value set items that are not set (blank items) are drawn using the background color of the value set.

```
param1
    USHORT  usRow      /* Row index. */
    USHORT  usColumn   /* Column index. */

param2
    ULONG   ulItemId   /* Item information. */
```

VM_SETITEM - Remarks

The application uses this message to set the contents of an individual value set item. To set the values for the entire value set, an application would loop through the rows and columns, setting the value of each item during the initial value set window processing before the window becomes visible.

VM_SETITEM - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

VM_SETITEM - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

VM_SETITEMATTR

VM_SETITEMATTR Field - usRow

usRow (USHORT)
Row index.

Row index of the value set item for which attributes are being specified. Rows have a value from 1 to the value of the *usRowCount* field. This value, which is the total number of rows in the value set, is specified in the **VSCDATA** data structure when the value set control is created.

VM_SETITEMATTR Field - usColumn

usColumn (USHORT)
Column index.

Column index of the value set item for which attributes are being specified. Columns have a value from 1 to the value of the *usColumnCount* field. This value, which is the total number of columns in the value set, is specified in the **VSCDATA** data structure when the value set control is created.

VM_SETITEMATTR Field - usItemAttr

usItemAttr (USHORT)
Item attributes.

Attribute or attributes of the item to be set or reset based on the value of the *fSet* field. These attributes can be as follows:

- One of the following attributes can be set:

VIA_BITMAP	If this attribute is set, the item is a bit map. This is the default.
VIA_COLORINDEX	If this attribute is set, the item is an index into the logical color table.
VIA_ICON	If this attribute is set, the item is an icon.
VIA_RGB	If this attribute is set, the item is a color entry.
VIA_TEXT	If this attribute is set, the item is a text string.
- In addition, one or more of the following attributes can be set:

VIA_DISABLED	If this attribute is set, the item cannot be selected and is displayed with unavailable-state emphasis, if possible. Unavailable text items are always displayed with unavailable-state emphasis, according to CUA guidelines; for items displayed as color, bit maps, and icons, it is the application's responsibility to determine the best way to show that these items are unavailable, if possible.
--------------	---

The selection cursor can be moved to an unavailable item by

using either the keyboard navigation keys or a pointing device. This allows a user to press the F1 key to find out why that item cannot be selected.

VIA_DRAGGABLE

If this attribute is set, the item can be the source of a direct manipulation action.

VIA_DROPONABLE

If this attribute is set, the item can be the target of a direct manipulation action.

VIA_OWNERDRAW

If this attribute is set, a paint notification message is sent whenever this item needs painting.

VM_SETITEMATTR Field - fSet

fSet ([USHORT](#))

Set or reset flag.

TRUE

Set the attribute of the indicated item.

FALSE

Turn off the attribute of the indicated item.

VM_SETITEMATTR Return Value - rc

rc ([BOOL](#))

Success indicator.

TRUE

Attribute or attributes were set successfully.

FALSE

An error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_PARAMETER_OUT_OF_RANGE.

VM_SETITEMATTR - Parameters

usRow ([USHORT](#))

Row index.

Row index of the value set item for which attributes are being specified. Rows have a value from 1 to the value of the *usRowCount* field. This value, which is the total number of rows in the value set, is specified in the [VSCDATA](#) data structure when the value set control is created.

usColumn ([USHORT](#))

Column index.

Column index of the value set item for which attributes are being specified. Columns have a value from 1 to the value of the *usColumnCount* field. This value, which is the total number of columns in the value set, is specified in the [VSCDATA](#) data structure when the value set control is created.

usItemAttr ([USHORT](#))
Item attributes.

Attribute or attributes of the item to be set or reset based on the value of the *fSet* field. These attributes can be as follows:

- One of the following attributes can be set:

VIA_BITMAP	If this attribute is set, the item is a bit map. This is the default.
VIA_COLORINDEX	If this attribute is set, the item is an index into the logical color table.
VIA_ICON	If this attribute is set, the item is an icon.
VIA_RGB	If this attribute is set, the item is a color entry.
VIA_TEXT	If this attribute is set, the item is a text string.
- In addition, one or more of the following attributes can be set:

VIA_DISABLED	If this attribute is set, the item cannot be selected and is displayed with unavailable-state emphasis, if possible. Unavailable text items are always displayed with unavailable-state emphasis, according to CUA guidelines; for items displayed as color, bit maps, and icons, it is the application's responsibility to determine the best way to show that these items are unavailable, if possible. The selection cursor can be moved to an unavailable item by using either the keyboard navigation keys or a pointing device. This allows a user to press the F1 key to find out why that item cannot be selected.
VIA_DRAGGABLE	If this attribute is set, the item can be the source of a direct manipulation action.
VIA_DROPONABLE	If this attribute is set, the item can be the target of a direct manipulation action.
VIA_OWNERDRAW	If this attribute is set, a paint notification message is sent whenever this item needs painting.

fSet ([USHORT](#))
Set or reset flag.

- | | |
|-------|---|
| TRUE | Set the attribute of the indicated item. |
| FALSE | Turn off the attribute of the indicated item. |

rc ([BOOL](#))
Success indicator.

- | | |
|-------|---|
| TRUE | Attribute or attributes were set successfully. |
| FALSE | An error occurred. The WinGetLastError function may return the following errors: <ul style="list-style-type: none">PMERR_INVALID_PARAMETERSPMERR_PARAMETER_OUT_OF_RANGE. |

VM_SETITEMATTR - Syntax

This message sets the attribute or attributes of the item indicated by the values of the *usRow* and *usColumn* parameters.

```
param1
    USHORT  usRow      /* Row index. */
    USHORT  usColumn   /* Column index. */

param2
    USHORT  usItemAttr /* Item attributes. */
    USHORT  fSet        /* Set or reset flag. */
```

VM_SETITEMATTR - Remarks

The application uses this message to either set or reset a specific attribute or attributes of a value set item. This provides customization of a control at the item level, so that applications can provide their own types of items with a value set, as well as perform direct manipulation and other actions.

VM_SETITEMATTR - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

VM_SETITEMATTR - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

VM_SETMETRICS

VM_SETMETRICS Field - fMetric

fMetric ([USHORT](#))
Units of measurement.

Unit or units of measurement that are to be set for the value set control. This can be either of the following:

VMA_ITEMSIZE

If this message attribute is set, the width and height of each item is set using the values of the *usItemWidth* and *usItemHeight* parameters, respectively.

VMA_ITEMSPACING

If this message attribute is set, the horizontal and vertical spacing between each item is set using the values of the *usHorzItemSpacing* and *usVertItemSpacing* parameters, respectively.

VM_SETMETRICS Field - ullItemId

ullItemId (ULONG)

Item information.

This value depends on the VMA_* attribute set for the message.

- If the VMA_ITEMSIZE attribute is specified, the *ullItemId* field is as follows:

usItemWidth (USHORT)

Width to be set for each value set item, in pixels. The number of pixels specified cannot be less than 2.

usItemHeight (USHORT)

Height to be set for each value set item, in pixels. The number of pixels specified cannot be less than 2.

- If the VMA_ITEMSPACING attribute is specified, *ullItemId* field is as follows:

usHorzItemSpacing (USHORT)

Amount of horizontal space to be set between each value set item, in pixels. This number does not include the space needed for selected-state and target emphasis, and for the selection cursor, because the emphasis and cursor space is automatically set by the value set control. The default spacing is 0.

usVertItemSpacing (USHORT)

Amount of vertical space to be set between each value set item, in pixels. This number does not include the space needed for selected-state and target emphasis, and for the selection cursor, because the emphasis and cursor space is automatically set by the value set control. The default spacing is 0.

VM_SETMETRICS Return Value - rc

rc (BOOL)

Success indicator.

TRUE

Item size or spacing was successfully set.

FALSE

An error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_PARAMETER_OUT_OF_RANGE.

VM_SETMETRICS - Parameters

fMetric (USHORT)

Units of measurement.

Unit or units of measurement that are to be set for the value set control. This can be either of the following:

VMA_ITEMSIZE

If this message attribute is set, the width and height of each item is set using the values of the *usItemWidth* and *usItemHeight* parameters, respectively.

VMA_ITEMSPACING

If this message attribute is set, the horizontal and vertical spacing between each item is set using the values of the *usHorzItemSpacing* and *usVertItemSpacing* parameters, respectively.

ulItemid (ULONG)

Item information.

This value depends on the VMA_* attribute set for the message.

- If the VMA_ITEMSIZE attribute is specified, the *ulItemid* field is as follows:

usItemWidth (USHORT)

Width to be set for each value set item, in pixels. The number of pixels specified cannot be less than 2.

usItemHeight (USHORT)

Height to be set for each value set item, in pixels. The number of pixels specified cannot be less than 2.

- If the VMA_ITEMSPACING attribute is specified, *ulItemid* field is as follows:

usHorzItemSpacing (USHORT)

Amount of horizontal space to be set between each value set item, in pixels. This number does not include the space needed for selected-state and target emphasis, and for the selection cursor, because the emphasis and cursor space is automatically set by the value set control. The default spacing is 0.

usVertItemSpacing (USHORT)

Amount of vertical space to be set between each value set item, in pixels. This number does not include the space needed for selected-state and target emphasis, and for the selection cursor, because the emphasis and cursor space is automatically set by the value set control. The default spacing is 0.

rc (BOOL)

Success indicator.

TRUE

Item size or spacing was successfully set.

FALSE

An error occurred. The [WinGetLastError](#) function may return the following errors:

- PMERR_INVALID_PARAMETERS
- PMERR_PARAMETER_OUT_OF_RANGE.

VM_SETMETRICS - Syntax

This message sets the size of each item in the value set control, the spacing between items, or both.

```
param1
    USHORT    fMetric    /* Units of measurement. */

param2
    ULONG     ulItemid   /* Item information. */
```

VM_SETMETRICS - Remarks

Upon receiving this message, the value set redraws the control with the new width, height, and spacing specifications for each item. Any items that do not fit within the current window size are clipped.

When the value set control receives a [WM_SIZE \(in Value Set Controls\)](#) message, which is sent when the value set window is resized, the value set control defaults the size of each item by dynamically dividing the window size by the number of rows and columns. It allows enough room for the border, selection cursor, and selection emphasis, and defaults the spacing between items to 0. To override these default settings, the application must resend the VM_SETMETRICS message.

VM_SETMETRICS - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE.

VM_SETMETRICS - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_CHAR (in Value Set Controls)

WM_CHAR (in Value Set Controls) - Syntax

For the cause of this message, see [WM_CHAR](#).
For a description of the parameters, see [WM_CHAR](#).

WM_CHAR (in Value Set Controls) - Remarks

The value set control window procedure responds to this message by sending it to its owner if it has not processed the key stroke. This is the most common means by which the focus is switched from one control to another in a value set window.

The keystrokes processed by a value set control are:

Key Name	Action Performed
Down Arrow	Moves the selection cursor down one item. When the selection cursor reaches the bottom, the Down Arrow has no effect.

Up Arrow	Moves the selection cursor up one item. When the selection cursor reaches the top, the Up Arrow has no effect.
Left Arrow	Moves the selection cursor left one item. When the selection cursor reaches the leftmost column, the Left Arrow has no effect.
Right Arrow	Moves the selection cursor right one item. When the selection cursor reaches the rightmost column, the Right Arrow has no effect.
Home	Moves the selection cursor to the leftmost column of the value set control (NLS dependent). Pressing the Home key when the leftmost column is selected has no effect. The row index does not change.
End	Moves the selection cursor to the rightmost column of the value set control (NLS dependent). Pressing the End key when the rightmost column is selected has no effect. The row index does not change.
PgDn	Moves the selection cursor to the bottom row of the value set control. Pressing the Page Down key when the bottom row is selected has no effect. The column index does not change.
PgUp	Moves the selection cursor to the top row of the value set control. Pressing the Page Up key when the top row is selected has no effect. The column index does not change.
Ctrl+Home	Moves the selection cursor to the item in the top row and leftmost column of the value set control (NLS dependent). Pressing the Ctrl+Home keys when the top row and leftmost column is selected has no effect.
Ctrl+End	Moves the selection cursor to the bottom row and rightmost column of the value set control (NLS dependent). Pressing the Ctrl+End keys when the bottom row and rightmost column is selected has no effect.
Enter	Sends a VN_ENTER notification code to the owner of the value set with the row and column indices of the selected item.
(Mnemonic)	If the VS_TEXT style bit is set for the value set, any mnemonics specified can be used to select an item.

WM_CHAR (in Value Set Controls) - Default Processing

For a description of the default processing, see [WM_CHAR](#).

WM_CHAR (in Value Set Controls) - Topics

Select an item:

[Syntax](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_PRESPARAMCHANGED (in Value Set Controls)

WM_PRESPARAMCHANGED (in Value Set Controls) Field -

attrtype

attrtype (ULONG)

Attribute type.

Presentation parameter attribute identity. The following presentation parameters are initialized by the value set control. The initial value of each is shown in the following list:

PP_FOREGROUND_COLOR or PP_FOREGROUND_COLOR_INDEX
Item foreground color; used when displaying text and bit maps. This color is initialized to SYSCLR_WINDOWTEXT.

PP_BACKGROUND_COLOR or PP_BACKGROUND_COLOR_INDEX
Value set background color; used for entire control as the background. This color is initialized to SYSCLR_WINDOW.

PP_HILITE_BACKGROUND_COLOR or PP_HILITE_BACKGROUND_COLOR_INDEX
Selection color; this is the color used for selected-state and target emphasis. This color is initialized to SYSCLR_HILITE_BACKGROUND.

PP_BORDER_COLOR or PP_BORDER_COLOR_INDEX
Value set and item border color. This color is initialized to SYSCLR_WINDOWFRAME.

WM_PRESPARAMCHANGED (in Value Set Controls) Field - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

WM_PRESPARAMCHANGED (in Value Set Controls) Return Value - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

WM_PRESPARAMCHANGED (in Value Set Controls) - Parameters

attrtype (ULONG)

Attribute type.

Presentation parameter attribute identity. The following presentation parameters are initialized by the value set control. The initial value of each is shown in the following list:

PP_FOREGROUND_COLOR or PP_FOREGROUND_COLOR_INDEX

Item foreground color; used when displaying text and bit maps. This color is initialized to SYSCLR_WINDOWTEXT.

PP_BACKGROUND_COLOR or PP_BACKGROUND_COLOR_INDEX

Value set background color; used for entire control as the background. This color is initialized to SYSCLR_WINDOW.

PP_HILITEBACKGROUND_COLOR or PP_HILITEBACKGROUND_COLOR_INDEX

Selection color; this is the color used for selected-state and target emphasis. This color is initialized to SYSCLR_HILITEBACKGROUND.

PP_BORDER_COLOR or PP_BORDER_COLOR_INDEX

Value set and item border color. This color is initialized to SYSCLR_WINDOWFRAME.

ulReserved (ULONG)

Reserved value, should be 0.

ulReserved (ULONG)

Reserved value, should be 0.

WM_PRESPARAMCHANGED (in Value Set Controls) - Syntax

For the cause of this message, see [WM_PRESPARAMCHANGED](#).

```
param1
    ULONG attrtype    /* Attribute type. */

param2
    ULONG ulReserved  /* Reserved value, should be 0. */
```

WM_PRESPARAMCHANGED (in Value Set Controls) - Remarks

The application uses this message to notify the value set that a given inherited presentation parameter has changed.

WM_PRESPARAMCHANGED (in Value Set Controls) - Default Processing

For a description of the default processing, see [WM_PRESPARAMCHANGED](#).

WM_PRESPARAMCHANGED (in Value Set Controls) -

Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

WM_QUERYWINDOWPARAMS (in Value Set Controls)

WM_QUERYWINDOWPARAMS (in Value Set Controls) Field - wndparams

wndparams ([PWNDPARAMS](#))

Pointer to a [WNDPARAMS](#) window parameter structure.

For a value set, the valid values for the *fsStatus* field are WPM_CBCTLDATA and WPM_CTLDATA.

The flags in the *fsStatus* field are cleared as each item is processed. If the call is successful, the *fsStatus* field is NULL. If any item has not been processed, the flag for that item is still set.

WM_QUERYWINDOWPARAMS (in Value Set Controls) Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_QUERYWINDOWPARAMS (in Value Set Controls) Return Value - rc

rc ([BOOL](#))

Success indicator.

TRUE

Successful operation.

FALSE

Error occurred.

WM_QUERYWINDOWPARAMS (in Value Set Controls) - Parameters

wndparams ([PWNDPARAMS](#))

Pointer to a [WNDPARAMS](#) window parameter structure.

For a value set, the valid values for the *fsStatus* field are WPM_CBCTLDATA and WPM_CTLDATA.

The flags in the *fsStatus* field are cleared as each item is processed. If the call is successful, the *fsStatus* field is NULL. If any item has not been processed, the flag for that item is still set.

ulReserved ([ULONG](#))

Reserved value, should be 0.

rc ([BOOL](#))

Success indicator.

TRUE

Successful operation.

FALSE

Error occurred.

WM_QUERYWINDOWPARAMS (in Value Set Controls) - Syntax

For the cause of this message, see [WM_QUERYWINDOWPARAMS](#).

```
param1
    PWNDPARAMS  wndparams    /* Pointer to a WNDPARAMS window parameter structure. */

param2
    ULONG         ulReserved  /* Reserved value, should be 0. */
```

WM_QUERYWINDOWPARAMS (in Value Set Controls) - Remarks

The value set control window procedure responds to this message by returning the information in the buffer provided. If this message is sent to a value set window of another process, the information in, or identified by, the *wndparams* parameter must be in memory shared by both processes.

WM_QUERYWINDOWPARAMS (in Value Set Controls) -

Default Processing

For a description of the default processing, see [WM_QUERYWINDOWPARAMS](#).

WM_QUERYWINDOWPARAMS (in Value Set Controls) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_SETWINDOWPARAMS (in Value Set Controls)

WM_SETWINDOWPARAMS (in Value Set Controls) Field - wndparams

wndparams ([PWNDPARAMS](#))

Pointer to a [WNDPARAMS](#) structure.

For a value set, the valid value of the *fsStatus* field is WPM_CTLDATA.

WM_SETWINDOWPARAMS (in Value Set Controls) Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_SETWINDOWPARAMS (in Value Set Controls) Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Successful operation
FALSE	Error occurred.

WM_SETWINDOWPARAMS (in Value Set Controls) - Parameters

wndparams ([PWNDPARAMS](#))
Pointer to a [WNDPARAMS](#) structure.

For a value set, the valid value of the *fsStatus* field is WPM_CTLDATA.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Success indicator.

TRUE	Successful operation
FALSE	Error occurred.

WM_SETWINDOWPARAMS (in Value Set Controls) - Syntax

For the cause of this message, see [WM_SETWINDOWPARAMS](#).

```
param1
    PWNDPARAMS  wndparams    /* Pointer to a WNDPARAMS structure. */

param2
    ULONG        ulReserved    /* Reserved value, should be 0. */
```

WM_SETWINDOWPARAMS (in Value Set Controls) - Remarks

If this message is sent to a value set window of another process, the information in, or identified by, the *wndparams* parameter must be in memory shared by both processes.

WM_SETWINDOWPARAMS (in Value Set Controls) - Default Processing

For a description of the default processing, see [WM_SETWINDOWPARAMS](#).

WM_SETWINDOWPARAMS (in Value Set Controls) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_SIZE (in Value Set Controls)

WM_SIZE (in Value Set Controls) - Syntax

For the cause of this message, see [WM_SIZE](#).

For a description of the parameters, see [WM_SIZE](#).

WM_SIZE (in Value Set Controls) - Remarks

When the value set window is sized, the value set control defaults the size of each item by dynamically dividing the window size by the number of rows and columns. It allows enough room for the border, selection cursor, and selection emphasis, and defaults the spacing between items to 0. To override these default settings, the application must resend the [VM_SETMETRICS](#) message.

WM_SIZE (in Value Set Controls) - Default Processing

For a description of the default processing, see [WM_SIZE](#).

WM_SIZE (in Value Set Controls) - Topics

Select an item:

[Syntax](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

Default Window Procedure Message Processing

This system-provided window procedure processes the actions that control the operation of windows.

Purpose

General window messages are used for standard processing. These messages can be requested from the system or sent to the system for information, or for actions such as create window, validate window, track mouse movement, and select and deselect actions.

Reserved Messages

These message ranges are reserved:

WM_USER All messages below this value are reserved for system use. Private messages must have an identifier with a value of WM_USER or higher.

Note: The operating system uses certain message values higher than WM_USER. These message values should not be used by an application. A partial listing of these messages is in the following figure:

From PMSTDDL.G.H:

```
#define FDM_FILTER           WM_USER+40
#define FDM_VALIDATE        WM_USER+41
#define FDM_ERROR           WM_USER+42

#define FNTM_FACENAMECHANGED WM_USER+50
#define FNTM_POINTSIZACHANGED WM_USER+51
#define FNTM_STYLECHANGED    WM_USER+52
#define FNTM_COLORCHANGED    WM_USER+53
#define FNTM_UPDATEPREVIEW   WM_USER+54
#define FNTM_FILTERLIST      WM_USER+55
```

You should scan your header files to see if other messages have been defined with values higher than WM_USER.

General Window Styles

The *window* is the mechanism by which the application communicates with the operator. Each window can have a window *style* that controls the appearance and behavior of the window. There are also *class* styles that apply to all the windows of a particular class (class being FRAME, BUTTON, and so on).

See [Window Class Styles](#) and [Window Styles](#) for more information.

Window Class Styles

These window class styles are available:

CS_SIZEREDRAW

Determines whether a window will be redrawn when sized. This style is to be used for a window whose contents are sensitive to the size of the window. For example, the data in some windows can be scaled up or down to fit the size of the Client Area. In other windows, the data remains the same size whatever the size of the window; it is merely clipped if the window is made smaller. The CS_SIZEREDRAW style is to be used in the first instance but not in the second. For more information, see [WM_CALCVAILIRECTS](#).

CS_SYNCPAINT

Window is synchronously repainted. This style causes WS_SYNCPAINT to be set for all windows of this class.

CS_MOVENOTIFY

This class style should be used by a child window if it wants to be notified with a [WM_MOVE](#) message when its parent is moved. For more detail, see the [WM_MOVE](#) message description.

CS_CLIPCHILDREN

Causes a window of style WS_CLIPCHILDREN to be created, regardless of whether this style bit is specified on the create window function.

CS_CLIPSIBLINGS

Causes a window of style WS_CLIPSIBLINGS to be created, regardless of whether this style bit is specified on the create window function.

CS_PARENTCLIP

Causes a window of style WS_PARENTCLIP to be created, regardless of whether this style bit is specified on the create window function.

CS_SAVEBITS

Causes a window of style WS_SAVEBITS to be created, regardless of whether this style bit is specified on the create window function.

CS_PUBLIC

Causes a public window class to be registered. It is an error if this parameter is specified on any process other than the shell process.

CS_HITTEST

If set, causes a [WM_HITTEST](#) message to be sent to the window, before sending any pointing device message.

If not set, no [WM_HITTEST](#) message is sent, and it is assumed that the window returns HT_NORMAL if the window is not disabled, and HT_ERROR if the window is disabled.

Top-level frame windows do not have CS_HITTEST set.

CS_FRAME

If set, all windows of this class are expected to behave as frame windows.

Window Styles

These window styles are available:

WS_SYNCPAINT

Window is synchronously repainted.

This style is set for windows that have Class Style CS_SYNCPAINT. Applications can then turn this style on and off to vary the window processing.

System-Provided Window Styles:

WS_ANIMATE

This specifies that window animation will be turned on. Windows animation is a visual effect that occurs when the window is opened or closed; the window seems to zoom out when it is opened, and zoom in when it is closed.

This visual effect also depends on the Animation setting in the System-Settings notebook. If Animation is enabled and this window style is set, window animation occurs when the window is opened or closed. When Animation is disabled in the System-Settings notebook, this style has no effect and no window animation occurs.

WS_CLIPCHILDREN

This specifies that the area occupied by the children of a window is to be excluded when drawing in that window. Normally, it is included.

WS_CLIPSIBLINGS

This specifies that the area occupied by the siblings of a window is to be excluded when drawing in that window. Normally, it is included.

WS_DISABLED

This specifies that the window is disabled. The default is enabled.

WS_MAXIMIZED

This specifies that the frame window is to be created maximized.

When a window is moved or sized in the normal way at least one border should remain on the screen. When a window is maximized and the maximum size is as large as the screen all borders should be positioned just outside the screen.

WS_MINIMIZED

This specifies that the frame window is to be created minimized.

WS_PARENTCLIP

This controls how a window is clipped when a drawing action takes place into the window.

Generally, a WS_PARENTCLIP window is not to draw outside its window rectangle.

WS_SAVEBITS

This specifies that the screen image of the area under a window of this style be saved when the window is made visible.

WS_VISIBLE

This specifies that the window is visible. The default is invisible.

Note: A window can still be visible, in this sense, even if it cannot be seen because it is covered by other windows.

Styles for Windows in Dialogs

WS_GROUP

This identifies the dialog items that make up a group.

This style is to be specified on the first window of any group. Subsequent windows of the group must not have this style. The windows of the group must be adjacent siblings. This can be done by listing the windows consecutively in [Dialog Template](#) or by inserting each new window in the group behind the previous one ([WinCreateWindow](#)).

WS_TABSTOP

This identifies a dialog item as one to which the operator can TAB.

General Window Messages

This section describes the window procedure actions upon receiving the following messages.

PL_ALTERED

PL_ALTERED Field - hiniUser

hiniUser ([HINI](#))

Handle of the new user profile.

PL_ALTERED Field - hiniSystem

hiniSystem ([HINI](#))
Handle of the new system profile.

PL_ALTERED Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, must be 0.

PL_ALTERED - Parameters

hiniUser ([HINI](#))
Handle of the new user profile.

hiniSystem ([HINI](#))
Handle of the new system profile.

ulReserved ([ULONG](#))
Reserved value, must be 0.

PL_ALTERED - Syntax

This message is broadcast to all frame windows when the [PrfReset](#) function is issued.

```
param1
    HINI    hiniUser    /* Handle of the new user profile. */

param2
    HINI    hiniSystem  /* Handle of the new system profile. */
```

PL_ALTERED - Remarks

Applications should refresh their defaults from the user or system profile.

PL_ALTERED - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

PL_ALTERED - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_ACTIVATE

WM_ACTIVATE Field - usactive

- usactive** ([USHORT](#))
Active indicator.
- | | |
|-------|----------------------------------|
| TRUE | The window is being activated |
| FALSE | The window is being deactivated. |

WM_ACTIVATE Field - hwnd

- hwnd** ([HWND](#))
Window handle.
- In the case of activation, *hwnd* identifies the window being activated. In the case of deactivation, *hwnd* identifies the window being deactivated.

WM_ACTIVATE Return Value - ulReserved

- ulReserved** ([ULONG](#))
Reserved value, should be 0.

WM_ACTIVATE - Parameters

usactive (USHORT)

Active indicator.

TRUE

The window is being activated

FALSE

The window is being deactivated.

hwnd (HWND)

Window handle.

In the case of activation, *hwnd* identifies the window being activated. In the case of deactivation, *hwnd* identifies the window being deactivated.

ulReserved (ULONG)

Reserved value, should be 0.

WM_ACTIVATE - Syntax

This message occurs when an application causes the activation or deactivation of a window.

```
param1
    USHORT    usactive    /* Active indicator. */

param2
    HWND      hwnd        /* Window handle. */
```

WM_ACTIVATE - Remarks

A deactivation message (that is, a WM_ACTIVATE message with *usactive* set to FALSE) is sent first to the window procedure of the main window being deactivated, before an activation message (that is, a WM_ACTIVATE message with *usactive* set to TRUE) is sent to the window procedure of the main window being activated.

Any WM_SETFOCUS messages with *usfocus* set to FALSE, are sent before the deactivation message. Any WM_SETFOCUS messages with *usfocus* set to TRUE, are sent after the activation message.

If WinSetFocus is called during the processing of a WM_ACTIVATE message, a WM_SETFOCUS message with *usfocus* set to FALSE is not sent, as no window has the focus.

If a window is activated before any of its children have the focus, this message is sent to the frame window or to its FID_CLIENT, if it exists.

Note: Except in the instance of a WM_ACTIVATE message, with *usactive* set to TRUE, an application processing a WM_ACTIVATE, or a WM_SETFOCUS message should not change the focus window or the active window. If it does, the focus and active windows must be restored before the window procedure returns from processing the message. For this reason, any dialog boxes or windows brought up during the processing of a WM_ACTIVATE, or a WM_SETFOCUS message should be system modal.

WM_ACTIVATE - Default Processing

The default window procedure takes no action on this message, other than to set *u/Reserved* to 0.

WM_ACTIVATE - Related Messages

Related Messages

- [WM_ACTIVATE \(in Frame Controls\)](#)
 - [WM_ACTIVATE \(Language Support Dialog\)](#)
 - [WM_ACTIVATE \(Language Support Window\)](#)
-

WM_ACTIVATE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_APPTERMINATENOTIFY

WM_APPTERMINATENOTIFY Field - happ

happ ([HAPP](#))
Application handle.

WM_APPTERMINATENOTIFY Field - flretcode

flretcode ([ULONG](#))
Return code from the terminating application.

WM_APPTERMINATENOTIFY Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, must be 0.

WM_APPTERMINATENOTIFY - Parameters

happ ([HAPP](#))
Application handle.

flretcode ([ULONG](#))
Return code from the terminating application.

ulReserved ([ULONG](#))
Reserved value, must be 0.

WM_APPTERMINATENOTIFY - Syntax

This message is posted when an application (started by another application) terminates.

```
param1
    HAPP    happ        /* Application handle. */

param2
    ULONG    flretcode    /* Return code from the terminating application. */
```

WM_APPTERMINATENOTIFY - Remarks

The WM_APPTERMINATENOTIFY message provides the capability for the starting application to be notified when the started application terminates.

WM_APPTERMINATENOTIFY - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_APPTERMINATENOTIFY - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

WM_ADJUSTWINDOWPOS

WM_ADJUSTWINDOWPOS Field - pswp

pswp ([PSWP](#))
SWP structure pointer.

The structure has been filled in by the [WinSetWindowPos](#) function with the proposed move or size data. The control can adjust this new position by changing the contents of the [SWP](#) structure. It can change the *x* or *y* fields to adjust its new position; or the *cx* or *cy* fields to adjust its new size, or the *hwndInsertBehind* field to adjust its new z-order.

WM_ADJUSTWINDOWPOS Field - flzero

flzero ([ULONG](#))
Zero.

WM_ADJUSTWINDOWPOS Return Value - flResult

flResult ([ULONG](#))
Window-adjustment status indicators.

These indicators are passed on to the [WM_WINDOWPOSCHANGED](#) message that is sent after the window state change has occurred. Bits 0 through 15 of this parameter are reserved for system use and bits 16 through 31 are available for application use.

- 0
No changes have been made
- AWP_MINIMIZED
The frame window has been minimized.
- AWP_MAXIMIZED

	The frame window has been maximized.
AWP_RESTORED	
	The frame window has been restored.
AWP_ACTIVATE	
	The frame window has been activated.
AWP_DEACTIVATE	
	The frame window has been deactivated.

WM_ADJUSTWINDOWPOS - Parameters

pswp (PSWP)

SWP structure pointer.

The structure has been filled in by the [WinSetWindowPos](#) function with the proposed move or size data. The control can adjust this new position by changing the contents of the [SWP](#) structure. It can change the *x* or *y* fields to adjust its new position; or the *cx* or *cy* fields to adjust its new size, or the *hwndInsertBehind* field to adjust its new z-order.

flzero (ULONG)

Zero.

flResult (ULONG)

Window-adjustment status indicators.

These indicators are passed on to the [WM_WINDOWPOSCHANGED](#) message that is sent after the window state change has occurred. Bits 0 through 15 of this parameter are reserved for system use and bits 16 through 31 are available for application use.

0	No changes have been made
AWP_MINIMIZED	
	The frame window has been minimized.
AWP_MAXIMIZED	
	The frame window has been maximized.
AWP_RESTORED	
	The frame window has been restored.
AWP_ACTIVATE	
	The frame window has been activated.
AWP_DEACTIVATE	
	The frame window has been deactivated.

WM_ADJUSTWINDOWPOS - Syntax

This message is sent by the [WinSetWindowPos](#) call to enable the window to adjust its new position or size whenever it is about to be moved.

```
param1
    PSWP    pswp        /* SWP structure pointer. */

param2
    ULONG   flzero       /* Zero. */
```

WM_ADJUSTWINDOWPOS - Remarks

Frame controls can respond to this message to reposition themselves or resize themselves in the window frame.

Menu controls respond to this message as follows:

MS_ACTIONBAR not specified

The [SWP cx](#) and [SWP cy](#) fields are set so that the menu window exactly contains all of the items in the menu. The [SWP x](#) and [SWP y](#) fields are not changed.

MS_ACTIONBAR specified and MS_TITLEBUTTON not specified

The items in the menu are arranged such that all of the items are visible within the width specified by the [SWP cx](#) field. This formatting may cause the menu items to be arranged in multiple lines. The [SWP cx](#) field is set to include all of the lines of the menu. The [SWP x](#) and [SWP y](#) fields are not changed.

MS_ACTIONBAR specified and MS_TITLEBUTTON specified

The [SWP cx](#) value is set to the accumulated width of the items in the menu. The height specified in the [SWP cy](#) field is not changed. In both instances, the [SWP cx](#) and [SWP cy](#) fields are only altered if SWP_SIZE is specified in the *//* field. Instead, the width of MS_TITLEBUTTON menus is determined by the accumulated width of the items in the menu.

A list box does two things:

- Changes the height so as to accommodate an exact number of items.
- Automatically outsides its border. This means, for example, that the x, y, width, and height fields in the resource file specify the working area of the listbox. The border is drawn outside this area.

The entry field control, if ES_MARGIN is specified, outsides its margin. This means that in the resource file, the numbers specified as the x-, and y-position of an entry field control are taken to be the position where the first character of text is drawn, not where the lower-left corner of the surrounding box is drawn. Similarly, the height and width parameters apply to the editable area of the control; consequently, they do not include the margin.

When a dialog is created with [WinCreateDlg](#) or [WinLoadDlg](#), a WM_ADJUSTWINDOWPOS message is sent to each child window after the dialog window is created, with a pointer to a [SWP](#) structure containing *//* equal to SWP_SIZE | SWP_MOVE and the *x*, *y*, *cy*, and *cx* fields initialized to the current size and position of the window. The message enables the control to adjust its size or position, usually to compensate for its border, or margin, or both.

WM_ADJUSTWINDOWPOS - Default Processing

The default window procedure takes no action on this message, other than to set *//Result* to 0.

WM_ADJUSTWINDOWPOS - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_BEGINDRAG

WM_BEGINDRAG Field - ptspointerpos

ptspointerpos (POINTS)
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *fPointer* is not set to TRUE.

WM_BEGINDRAG Field - fPointer

fPointer (USHORT)
Input device flag.

- | | |
|-------|---------------------------------------|
| TRUE | Message resulted from pointer event |
| FALSE | Message resulted from keyboard event. |

WM_BEGINDRAG Parameter - rc

rc (BOOL)
Processed indicator.

- | | |
|-------|-------------------|
| TRUE | Message processed |
| FALSE | Message ignored. |

WM_BEGINDRAG - Parameters

ptspointerpos (POINTS)
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *fPointer* is not set to TRUE.

fPointer (USHORT)
Input device flag.

- | | |
|-------|---------------------------------------|
| TRUE | Message resulted from pointer event |
| FALSE | Message resulted from keyboard event. |

rc (BOOL)

Processed indicator.

TRUE

Message processed

FALSE

Message ignored.

WM_BEGINDRAG - Syntax

This message occurs when the operator initiates a drag operation.

```
param1
    POINTS   ptspointerpos /* Pointer position. */
param2
    USHORT   fPointer      /* Input device flag. */
```

WM_BEGINDRAG - Remarks

This message is posted to the application queue associated with the window that has the focus, or with the window that is to receive the pointer-button information. This message will result from a mouse event, specified by the system value SV_BEGINDRAG.

WM_BEGINDRAG - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set result to FALSE.

WM_BEGINDRAG - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

WM_BEGINSELECT

WM_BEGINSELECT Field - ptspointerpos

ptspointerpos (POINTS)
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *fPointer* is not set to TRUE.

WM_BEGINSELECT Field - fPointer

fPointer (USHORT)
Input device flag.

TRUE	Message resulted from pointer event
FALSE	Message resulted from keyboard event.

WM_BEGINSELECT Return Value - rc

rc (BOOL)
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_BEGINSELECT - Parameters

ptspointerpos (POINTS)
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *fPointer* is not set to TRUE.

fPointer (USHORT)
Input device flag.

TRUE	Message resulted from pointer event
FALSE	Message resulted from keyboard event.

rc (BOOL)

Processed indicator.

TRUE

Message processed

FALSE

Message ignored.

WM_BEGINSELECT - Syntax

This message occurs when the operator initiates a swipe selection.

```
param1
    POINTS  ptspointerpos /* Pointer position. */

param2
    USHORT  fPointer      /* Input device flag. */
```

WM_BEGINSELECT - Remarks

This message is posted to the application queue associated with the window that has the focus, or with the window that is to receive the pointer-button information. This message will result from a mouse event, specified by the system value SV_BEGINSELECT.

WM_BEGINSELECT - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set result to FALSE.

WM_BEGINSELECT - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

WM_BUTTON1CLICK

WM_BUTTON1CLICK Field - ptspointerpos

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

WM_BUTTON1CLICK Field - fsHitTestres

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

WM_BUTTON1CLICK Field - fsflags

fsflags ([USHORT](#))
Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.
---------	-----------------------------------

WM_BUTTON1CLICK Return Value - rc

rc ([BOOL](#))
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_BUTTON1CLICK - Parameters

ptspointerpos ([POINTS](#))

Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

fsHitTestres ([USHORT](#))

Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

fsflags ([USHORT](#))

Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.
---------	-----------------------------------

rc ([BOOL](#))

Processed indicator.

TRUE	Message processed
------	-------------------

FALSE	Message ignored.
-------	------------------

WM_BUTTON1CLICK - Syntax

This message occurs when the operator presses and then releases button 1 of the pointing device within a specified period of time, and without moving the mouse.

```
param1
    POINTS    ptspointerpos    /* Pointer position. */

param2
    USHORT    fsHitTestres     /* Hit-test result. */
    USHORT    fsflags          /* Keyboard control codes. */
```

WM_BUTTON1CLICK - Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

WM_BUTTON1CLICK - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *rc* to FALSE.

WM_BUTTON1CLICK - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

WM_BUTTON1DBLCLK

WM_BUTTON1DBLCLK Field - ptspointerpos

ptspointerpos ([POINTS](#))

Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

WM_BUTTON1DBLCLK Field - fsHitTestres

fsHitTestres ([USHORT](#))

Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

WM_BUTTON1DBLCLK Field - fsflags

fsflags ([USHORT](#))

Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE

Indicates that no key is pressed.

WM_BUTTON1DBLCLK Return Value - rc

rc ([BOOL](#))
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_BUTTON1DBLCLK - Parameters

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

fsflags ([USHORT](#))
Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.
---------	-----------------------------------

rc ([BOOL](#))
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_BUTTON1DBLCLK - Syntax

This message occurs when the operator presses button 1 of the pointing device twice within a specified time, as detailed below.

```
param1
    POINTS  ptspointerpos  /*  Pointer position.  */

param2
    USHORT  fsHitTestres   /*  Hit-test result.  */
    USHORT  fsflags        /*  Keyboard control codes.  */
```

WM_BUTTON1DBLCLK - Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

A double-click is recognized if all of the following are true:

- Two clicks are of the same button.
- No intervening pointing device button is pressed.
- The two clicks occur within the double-click time interval as defined by the SV_DBLCLKTIME system value.
- The two clicks occur within a small spatial distance. This is defined by the rectangle, the length of whose sides parallel to the x- and y-axes are respectively, the SV_CXDBLCLICK and SV_CYDBLCLICK system values. The first click is assumed to be at the center of this rectangle.

The keyboard control codes specified by "flags" reflects the keyboard state at the time the mouse message was initiated. This may or may not reflect the current keyboard state.

WM_BUTTON1DBLCLK - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *rc* to FALSE.

WM_BUTTON1DBLCLK - Related Messages

Related Messages

- [WM_BUTTON1DBLCLK \(in Frame Controls\)](#)
- [WM_BUTTON1DBLCLK \(in Multiline Entry Fields\)](#)

WM_BUTTON1DBLCLK - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_BUTTON1DOWN

WM_BUTTON1DOWN Field - ptspointerpos

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

WM_BUTTON1DOWN Field - fsHitTestres

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit test process, which determined the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

WM_BUTTON1DOWN Field - fsflags

fsflags ([USHORT](#))
Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.
---------	-----------------------------------

WM_BUTTON1DOWN Return Value - rc

rc ([BOOL](#))
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_BUTTON1DOWN - Parameters

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit test process, which determined the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

fsflags (USHORT)

Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.
---------	-----------------------------------

rc (BOOL)

Processed indicator.

TRUE

Message processed

FALSE

Message ignored.

WM_BUTTON1DOWN - Syntax

This message occurs when the operator presses pointer button one.

```
param1
    POINTS    ptspointerpos /* Pointer position. */

param2
    USHORT    fsHitTestres /* Hit-test result. */
    USHORT    fsflags      /* Keyboard control codes. */
```

WM BUTTON1DOWN - Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

It is the responsibility of the application to ensure that the appropriate frame window is activated and that the focus is to the appropriate window, by using the [WinSetFocus](#) function. The keyboard control codes specified by "flags" reflects the keyboard state at the time the mouse message was initiated. This may or may not reflect the current keyboard state.

WM_BUTTON1DOWN - Default Processing

The default window procedure activates the window using [WinSetActiveWindow](#), and then sets *rc* to FALSE.

WM_BUTTON1DOWN - Related Messages

Related Messages

- [WM_BUTTON1DOWN](#) (in Frame Controls)
- [WM_BUTTON1DOWN](#) (in Multiline Entry Fields)

WM_BUTTON1DOWN - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Related Messages](#)

[Glossary](#)

WM_BUTTON1MOTIONEND

WM_BUTTON1MOTIONEND Field - ptspointerpos

ptspointerpos ([POINTS](#))

Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the hit-tested window, when the drag operation is terminated.

WM_BUTTON1MOTIONEND Field - fsHitTestres

fsHitTestres ([USHORT](#))

Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

WM_BUTTON1MOTIONEND Return Value - rc

rc ([BOOL](#))

Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_BUTTON1MOTIONEND - Parameters

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the hit-tested window, when the drag operation is terminated.

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

rc ([BOOL](#))
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_BUTTON1MOTIONEND - Syntax

This message occurs when the operator completes a drag operation which was initiated by pressing button one on the pointing device.

```
param1
    POINTS    ptspointerpos    /* Pointer position. */

param2
    USHORT    fsHitTestres     /* Hit-test result. */
```

WM_BUTTON1MOTIONEND - Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

WM_BUTTON1MOTIONEND - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than

to set *rc* to FALSE.

WM_BUTTON1MOTIONEND - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

WM_BUTTON1MOTIONSTART

WM_BUTTON1MOTIONSTART Field - ptspointerpos

ptspointerpos ([POINTS](#))

Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the hit-tested window, when the drag operation is started.

WM_BUTTON1MOTIONSTART Field - fsHitTestres

fsHitTestres ([USHORT](#))

Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

WM_BUTTON1MOTIONSTART Return Value - rc

rc ([BOOL](#))

Processed indicator.

- | | |
|-------|-------------------|
| TRUE | Message processed |
| FALSE | |

Message ignored.

WM_BUTTON1MOTIONSTART - Parameters

ptspointerpos ([POINTS](#))

Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the hit-tested window, when the drag operation is started.

fsHitTestres ([USHORT](#))

Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

rc ([BOOL](#))

Processed indicator.

TRUE

Message processed

FALSE

Message ignored.

WM_BUTTON1MOTIONSTART - Syntax

This message occurs when the operator initiates a drag operation by moving the mouse while pressing button one on the pointing device.

```
param1
    POINTS    ptspointerpos    /* Pointer position. */

param2
    USHORT    fsHitTestres     /* Hit-test result. */
```

WM_BUTTON1MOTIONSTART - Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

WM_BUTTON1MOTIONSTART - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *rc* to FALSE.

WM_BUTTON1MOTIONSTART - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

WM_BUTTON1UP

WM_BUTTON1UP Field - ptspointerpos

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

WM_BUTTON1UP Field - fsHitTestres

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

WM_BUTTON1UP Field - fsflags

fsflags ([USHORT](#))
Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.
---------	-----------------------------------

WM_BUTTON1UP Return Value - rc

rc (BOOL)	Processed indicator.
TRUE	Message processed
FALSE	Message ignored.

WM_BUTTON1UP - Parameters

ptspointerpos (POINTS)	Pointer position.
The pointer position is in window coordinates relative to the bottom-left corner of the window.	
fsHitTestres (USHORT)	Hit-test result.
<i>fsHitTestres</i> provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see WM_HITTEST .	
fsflags (USHORT)	Keyboard control codes.
In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.	
KC_NONE	Indicates that no key is pressed.
rc (BOOL)	Processed indicator.
TRUE	Message processed
FALSE	Message ignored.

WM_BUTTON1UP - Syntax

This message occurs when the operator releases button 1 of the pointing device.

```
param1
    POINTS  ptspointerpos /* Pointer position. */

param2
    USHORT  fsHitTestres /* Hit-test result. */
    USHORT  fsflags      /* Keyboard control codes. */
```

WM_BUTTON1UP - Remarks

This message is posted to the application queue associated with the window that is to receive the pointing device button information. The keyboard control codes specified by "flags" reflects the keyboard state at the time the mouse message was initiated. This may or may not reflect the current keyboard state.

WM_BUTTON1UP - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message other than to set *rc* to FALSE.

WM_BUTTON1UP - Related Messages

Related Messages

- [WM_BUTTON1UP \(in Frame Controls\)](#)
 - [WM_BUTTON1UP \(in Multiline Entry Fields\)](#)
-

WM_BUTTON1UP - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_BUTTON2CLICK

WM_BUTTON2CLICK Field - ptspointerpos

ptspointerpos (POINTS)

Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

WM_BUTTON2CLICK Field - fsHitTestres

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

WM_BUTTON2CLICK Field - fsflags

fsflags ([USHORT](#))
Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.
---------	-----------------------------------

WM_BUTTON2CLICK Return Value - rc

rc ([BOOL](#))
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_BUTTON2CLICK - Parameters

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

fsflags ([USHORT](#))
Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.
---------	-----------------------------------

rc ([BOOL](#))

Processed indicator.

TRUE	
------	--

Message processed

FALSE	
-------	--

Message ignored.

WM_BUTTON2CLICK - Syntax

This message occurs when the operator presses and then releases button 2 of the pointing device within a specified period of time, and without moving the mouse.

```
param1
    POINTS    ptspointerpos    /* Pointer position. */

param2
    USHORT    fsHitTestres     /* Hit-test result. */
    USHORT    fsflags          /* Keyboard control codes. */
```

WM_BUTTON2CLICK - Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

WM_BUTTON2CLICK - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set rc to FALSE.

WM_BUTTON2CLICK - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

WM_BUTTON2DBLCLK

WM_BUTTON2DBLCLK Field - ptspointerpos

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

WM_BUTTON2DBLCLK Field - fsHitTestres

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

WM_BUTTON2DBLCLK Field - fsflags

fsflags ([USHORT](#))
Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.
---------	-----------------------------------

WM_BUTTON2DBLCLK Return Value - rc

rc ([BOOL](#))
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_BUTTON2DBLCLK - Parameters

ptspointerpos (POINTS)

Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

fsHitTestres (USHORT)

Hit-test result.

IsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

fsflags (USHORT)

Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.
---------	-----------------------------------

rc (BOOL)

Processed indicator.

TRUE

Message processed

FALSE

Message ignored.

WM_BUTTON2DBLCLK - Syntax

This message occurs when the operator presses button 2 of the pointing device twice within a specified time, as detailed in [WM BUTTON1DBLCLK](#).

```
param1
    POINTS    ptspointerpos    /* Pointer position. */

param2
    USHORT    fsHitTestres     /* Hit-test result. */
    USHORT    fsflags          /* Keyboard control codes. */
```

WM_BUTTON2DBLCLK - Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information. The keyboard control codes specified by "flags" reflects the keyboard state at the time the mouse message was initiated. This may or may not reflect the current keyboard state.

WM_BUTTON2DBLCLK - Default Processing

The default window procedure processes this message identically to [WM_BUTTON1DBLCLK](#).

WM_BUTTON2DBLCLK - Related Messages

Related Messages

- [WM_BUTTON2DBLCLK](#) (in [Frame Controls](#))
-

WM_BUTTON2DBLCLK - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_BUTTON2DOWN

WM_BUTTON2DOWN Field - ptspointerpos

ptspointerpos ([POINTS](#))

Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

WM_BUTTON2DOWN Field - fsHitTestres

fsHitTestres ([USHORT](#))

Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit test process, which determined the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

WM_BUTTON2DOWN Field - fsflags

fsflags (USHORT)

Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.
---------	-----------------------------------

WM_BUTTON2DOWN Return Value - rc

rc (BOOL)

Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_BUTTON2DOWN - Parameters

ptspointerpos (POINTS)

Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

fsHitTestres (USHORT)

Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit test process, which determined the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

fsflags (USHORT)

Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.
---------	-----------------------------------

rc (BOOL)

Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_BUTTON2DOWN - Syntax

This message occurs when the operator presses button 2 on the pointing device.

```
param1
    POINTS    ptspointerpos    /* Pointer position. */

param2
    USHORT    fsHitTestres     /* Hit-test result. */
    USHORT    fsflags          /* Keyboard control codes. */
```

WM_BUTTON2DOWN - Remarks

This message is posted to the application queue associated with the window that is to receive the pointing device button information.

It is the responsibility of the application to ensure that the appropriate frame window is activated and that the focus is to the appropriate window, by using the [WinSetFocus](#) function. The keyboard control codes specified by "flags" reflects the keyboard state at the time the mouse message was initiated. This may or may not reflect the current keyboard state.

WM_BUTTON2DOWN - Default Processing

The default window procedure processes this message identically to [WM_BUTTON1DOWN](#).

WM_BUTTON2DOWN - Related Messages

Related Messages

- [WM_BUTTON2DOWN](#) (in Frame Controls)

WM_BUTTON2DOWN - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_BUTTON2MOTIONEND

WM_BUTTON2MOTIONEND Field - ptspointerpos

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the hit-tested window, when the drag operation is terminated.

WM_BUTTON2MOTIONEND Field - fsHitTestres

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

WM_BUTTON2MOTIONEND Return Value - rc

rc ([BOOL](#))
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_BUTTON2MOTIONEND - Parameters

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the hit-tested window, when the drag operation is terminated.

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

rc ([BOOL](#))

Processed indicator.

TRUE

Message processed

FALSE

Message ignored.

WM_BUTTON2MOTIONEND - Syntax

This message occurs when the operator completes a drag operation which was initiated by pressing button two on the pointing device.

```
param1
    POINTS   ptspointerpos /* Pointer position. */
param2
    USHORT   fsHitTestres /* Hit-test result. */
```

WM_BUTTON2MOTIONEND - Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

WM_BUTTON2MOTIONEND - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *rc* to FALSE.

WM_BUTTON2MOTIONEND - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

WM_BUTTON2MOTIONSTART

WM_BUTTON2MOTIONSTART Field - ptspointerpos

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the hit-tested window, when the drag operation is started.

WM_BUTTON2MOTIONSTART Field - fsHitTestres

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

WM_BUTTON2MOTIONSTART Return Value - rc

rc ([BOOL](#))
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_BUTTON2MOTIONSTART - Parameters

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the hit-tested window, when the drag operation is started.

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

rc ([BOOL](#))
Processed indicator.

TRUE	Message processed
------	-------------------

FALSE

Message ignored.

WM_BUTTON2MOTIONSTART - Syntax

This message occurs when the operator initiates a drag operation by moving the mouse while pressing button two on the pointing device.

```
param1
    POINTS  ptspointerpos /* Pointer position. */
param2
    USHORT  fsHitTestres /* Hit-test result. */
```

WM_BUTTON2MOTIONSTART - Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

WM_BUTTON2MOTIONSTART - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *rc* to FALSE.

WM_BUTTON2MOTIONSTART - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_BUTTON2UP

WM_BUTTON2UP Field - ptspointerpos

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

WM_BUTTON2UP Field - fsHitTestres

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

WM_BUTTON2UP Field - fsflags

fsflags ([USHORT](#))
Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.
---------	-----------------------------------

WM_BUTTON2UP Return Value - rc

rc ([BOOL](#))
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_BUTTON2UP - Parameters

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

fsflags (USHORT)

Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.
---------	-----------------------------------

rc (BOOL)

Processed indicator.

TRUE

Message processed

FALSE

Message ignored.

WM_BUTTON2UP - Syntax

This message occurs when the operator releases button 2 of the pointing device.

```
param1
    POINTS    ptspointerpos /* Pointer position. */

param2
    USHORT    fsHitTestres /* Hit-test result. */
    USHORT    fsflags      /* Keyboard control codes. */
```

WM_BUTTON2UP - Remarks

This message is posted to the application queue associated with the window that is to receive the pointing device button information. The keyboard control codes specified by "flags" reflects the keyboard state at the time the mouse message was initiated. This may or may not reflect the current keyboard state.

WM_BUTTON2UP - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message other than to set `rc` to FALSE.

WM_BUTTON2UP - Related Messages

Related Messages

- WM_BUTTON2UP (in Frame Controls)

WM_BUTTON2UP - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Related Messages](#)

[Glossary](#)

WM_BUTTON3CLICK

WM_BUTTON3CLICK Field - ptspointerpos

ptspointerpos ([POINTS](#))

Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

WM_BUTTON3CLICK Field - fsHitTestres

fsHitTestres ([USHORT](#))

Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

WM_BUTTON3CLICK Field - fsflags

fsflags ([USHORT](#))

Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE

Indicates that no key is pressed.

WM_BUTTON3CLICK Return Value - rc

rc (**BOOL**)

Processed indicator.

TRUE

Message processed

FALSE

Message ignored.

WM_BUTTON3CLICK - Parameters

ptspointerpos (**POINTS**)

Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

fsHitTestres (**USHORT**)

Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

fsflags (**USHORT**)

Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE

Indicates that no key is pressed.

rc (**BOOL**)

Processed indicator.

TRUE

Message processed

FALSE

Message ignored.

WM_BUTTON3CLICK - Syntax

This message occurs when the operator presses and then releases button 3 of the pointing device within a specified period of time, and without moving the mouse.

```
param1
    POINTS    ptspointerpos    /* Pointer position. */

param2
    USHORT    fsHitTestres     /* Hit-test result. */
```

```
USHORT fsflags /* Keyboard control codes. */
```

WM_BUTTON3CLICK - Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

WM_BUTTON3CLICK - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *rc* to FALSE.

WM_BUTTON3CLICK - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_BUTTON3DBLCLK

WM_BUTTON3DBLCLK Field - ptspinterpos

ptspinterpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom left corner of the window.

WM_BUTTON3DBLCLK Field - fsHitTestres

fsHitTestres ([USHORT](#))

Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

WM_BUTTON3DBLCLK Field - fsflags

fsflags ([USHORT](#))

Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.
---------	-----------------------------------

WM_BUTTON3DBLCLK Return Value - rc

rc ([BOOL](#))

Processed indicator.

TRUE	
	Message processed
FALSE	
	Message ignored.

WM_BUTTON3DBLCLK - Parameters

ptspointerpos ([POINTS](#))

Pointer position.

The pointer position is in window coordinates relative to the bottom left corner of the window.

fsHitTestres ([USHORT](#))

Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

fsflags ([USHORT](#))

Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.
---------	-----------------------------------

rc ([BOOL](#))

Processed indicator.

TRUE	
	Message processed
FALSE	
	Message ignored.

WM_BUTTON3DBLCLK - Syntax

This message occurs when the operator presses button 3 of the pointing device twice within a specified time, as detailed in [WM_BUTTON1DBLCLK](#).

```
param1
    POINTS    ptspointerpos /* Pointer position. */

param2
    USHORT    fsHitTestres /* Hit-test result. */
    USHORT    fsflags      /* Keyboard control codes. */
```

WM_BUTTON3DBLCLK - Remarks

This message is posted to the application queue associated with the window that is to receive the pointer button information. The keyboard control codes specified by "flags" reflects the keyboard state at the time the mouse message was initiated. This may or may not reflect the current keyboard state.

WM_BUTTON3DBLCLK - Default Processing

The default window procedure processes this message identically to [WM_BUTTON1DBLCLK](#).

WM_BUTTON3DBLCLK - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_BUTTON3DOWN

WM_BUTTON3DOWN Field - ptspointerpos

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

WM_BUTTON3DOWN Field - fsHitTestres

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit test process, which determined the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

WM_BUTTON3DOWN Field - fsflags

fsflags ([USHORT](#))
Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.
---------	-----------------------------------

WM_BUTTON3DOWN Return Value - rc

rc ([BOOL](#))
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_BUTTON3DOWN - Parameters

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit test process, which determined the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

fsflags ([USHORT](#))

Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.
---------	-----------------------------------

rc ([BOOL](#))

Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_BUTTON3DOWN - Syntax

This message occurs when the operator presses button 3 on the pointing device.

```
param1
    POINTS    ptspointerpos    /* Pointer position. */

param2
    USHORT    fsHitTestres     /* Hit-test result. */
    USHORT    fsflags          /* Keyboard control codes. */
```

WM_BUTTON3DOWN - Remarks

This message is posted to the application queue associated with the window that is to receive the pointing device button information.

It is the responsibility of the application to ensure that the appropriate frame window is activated and that the focus is to the appropriate window, by using the [WinSetFocus](#) function. The keyboard control codes specified by "flags" reflects the keyboard state at the time the mouse message was initiated. This may or may not reflect the current keyboard state.

WM_BUTTON3DOWN - Default Processing

The default window procedure processes this message identically to [WM_BUTTON1DOWN](#).

WM_BUTTON3DOWN - Topics

Select an item:

[Syntax](#)

WM_BUTTON3MOTIONEND

WM_BUTTON3MOTIONEND Field - ptspointerpos

ptspointerpos ([POINTS](#))

Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the hit-tested window, when the drag operation is terminated.

WM_BUTTON3MOTIONEND Field - fsHitTestres

fsHitTestres ([USHORT](#))

Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

WM_BUTTON3MOTIONEND Return Value - rc

rc ([BOOL](#))

Processed indicator.

TRUE

Message processed

FALSE

Message ignored.

WM_BUTTON3MOTIONEND - Parameters

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the hit-tested window, when the drag operation is terminated.

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

rc ([BOOL](#))
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_BUTTON3MOTIONEND - Syntax

This message occurs when the operator completes a drag operation which was initiated by pressing button three on the pointing device.

```
param1
    POINTS    ptspointerpos    /*  Pointer position. */

param2
    USHORT    fsHitTestres     /*  Hit-test result. */
```

WM_BUTTON3MOTIONEND - Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

WM_BUTTON3MOTIONEND - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *rc* to FALSE.

WM_BUTTON3MOTIONEND - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)

WM_BUTTON3MOTIONSTART

WM_BUTTON3MOTIONSTART Field - ptspointerpos

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the hit-tested window, when the drag operation is started.

WM_BUTTON3MOTIONSTART Field - fsHitTestres

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

WM_BUTTON3MOTIONSTART Return Value - rc

rc ([BOOL](#))
Processed indicator.

- | | |
|-------|-------------------|
| TRUE | Message processed |
| FALSE | Message ignored. |
-

WM_BUTTON3MOTIONSTART - Parameters

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the hit-tested window, when the drag operation is started.

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

rc ([BOOL](#))
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_BUTTON3MOTIONSTART - Syntax

This message occurs when the operator initiates a drag operation by moving the mouse while pressing button three on the pointing device.

```
param1
    POINTS   ptspointerpos /* Pointer position. */

param2
    USHORT   fsHitTestres /* Hit-test result. */
```

WM_BUTTON3MOTIONSTART - Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

WM_BUTTON3MOTIONSTART - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *rc* to FALSE.

WM_BUTTON3MOTIONSTART - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

WM_BUTTON3UP

WM_BUTTON3UP Field - ptspointerpos

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

WM_BUTTON3UP Field - fsHitTestres

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

WM_BUTTON3UP Field - fsflags

fsflags ([USHORT](#))
Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed.
---------	-----------------------------------

WM_BUTTON3UP Return Value - rc

rc ([BOOL](#))
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

Hit-test result.

fshHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE Indicates that no key is pressed.

Processed indicator.

TRUE

Message processed

FALSE

Message ignored.

This message occurs when the operator releases button 3 of the pointing device.

```
param1
    POINTS    ptspointerpos    /* Pointer position. */

param2
    USHORT    fsHitTestres     /* Hit-test result. */
    USHORT    fsflags          /* Keyboard control codes. */
```

This message is posted to the application queue associated with the window that is to receive the pointing device button information. The keyboard control codes specified by "flags" reflects the keyboard state at the time the mouse message was initiated. This may or may not reflect the current keyboard state.

The default window procedure processes this message identically to [WM_BUTTON1UP](#).

WM_BUTTON3UP - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_CALCFRAMERECT

WM_CALCFRAMERECT Field - pRect

pRect ([PRECTL](#))
Rectangle structure.

This points to a [RECTL](#) structure.

WM_CALCFRAMERECT Field - usFrame

usFrame ([USHORT](#))
Frame indicator.

TRUE	Frame rectangle provided
FALSE	Client area rectangle provided.

WM_CALCFRAMERECT Return Value - rc

rc ([BOOL](#))
Rectangle-calculated indicator.

TRUE	Successful completion
FALSE	Error occurred or the calculated rectangle is empty.

WM_CALCFRAMERECT - Parameters

pRect ([PRECTL](#))
Rectangle structure.

This points to a [RECTL](#) structure.

usFrame ([USHORT](#))
Frame indicator.

TRUE	Frame rectangle provided
FALSE	Client area rectangle provided.

rc ([BOOL](#))
Rectangle-calculated indicator.

TRUE	Successful completion
FALSE	Error occurred or the calculated rectangle is empty.

WM_CALCFRAMERECT - Syntax

This message occurs when an application uses the [WinCalcFrameRect](#) function.

```
param1
    PRECTL pRect    /* Rectangle structure. */

param2
    USHORT usFrame  /* Frame indicator. */
```

WM_CALCFRAMERECT - Remarks

This message is sent to the frame control to perform the appropriate calculation. If the low word of MP2 is TRUE, the RECTL structure in MP1 contains a frame window and this message calculates the RECTL of the client. If the low word of MP2 is FALSE, MP1 contains a client window and this message calculates the RECTL of the frame.

WM_CALCFRAMERECT - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_CALCFRAMERECT - Related Messages

Related Messages

- [WM_CALCFRAMERECT](#) (in [Frame Controls](#))
-

WM_CALCFRAMERECT - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_CALCVALIDRECTS

WM_CALCVALIDRECTS Field - pOldNew

pOldNew ([PRECTL](#))

Window-rectangle structures.

This points to two [RECTL](#) structures. The first structure contains the rectangle of the window before the move, the second contains the rectangle of the window after the move. The coordinates of the rectangles are relative to the parent window.

WM_CALCVALIDRECTS Field - pNew

pNew ([PSWP](#))

New window position.

This points to a [SWP](#) structure that contains information about the window after it is resized (see the [WinSetWindowPos](#) function).

WM_CALCVALIDRECTS Return Value - usAlign

usAlign (USHORT)

Alignment control.

This instructs [WinSetWindowPos](#) how to align valid window bits. This value is made up from CVR_* flags, as follows:

CVR_ALIGNLEFT

Align with the left edge of the window.

CVR_ALIGNBOTTOM

Align with the bottom edge of the window.

CVR_ALIGNTOP

Align with the top edge of the window.

CVR_ALIGNRIGHT

Align with the right edge of the window.

CVR_REDRAW

The whole window is invalid. If CVR_REDRAW, is set, the whole window is assumed invalid, otherwise, the remaining flags can be ORed together to get different kinds of alignment. For example:

`(CVR_ALIGNLEFT | CVR_ALIGNTOP)`

aligns the valid window area with the top-left of the window.

0

It is assumed the application has changed the rectangles pointed to by *pOldNew* and *pNew* itself.

WM_CALCVALIDRECTS - Parameters

pOldNew (PRECTL)

Window-rectangle structures.

This points to two [RECTL](#) structures. The first structure contains the rectangle of the window before the move, the second contains the rectangle of the window after the move. The coordinates of the rectangles are relative to the parent window.

pNew (PSWP)

New window position.

This points to a [SWP](#) structure that contains information about the window after it is resized (see the [WinSetWindowPos](#) function).

usAlign (USHORT)

Alignment control.

This instructs [WinSetWindowPos](#) how to align valid window bits. This value is made up from CVR_* flags, as follows:

CVR_ALIGNLEFT

Align with the left edge of the window.

CVR_ALIGNBOTTOM

Align with the bottom edge of the window.

CVR_ALIGNTOP

Align with the top edge of the window.

CVR_ALIGNRIGHT

Align with the right edge of the window.

CVR_REDRAW

The whole window is invalid. If CVR_REDRAW, is set, the whole window is assumed invalid, otherwise, the remaining flags can be ORed together to get different kinds of alignment. For example:

```
(CVR_ALIGNLEFT | CVR_ALIGNTOP)
```

aligns the valid window area with the top-left of the window.

0

It is assumed the application has changed the rectangles pointed to by *pOldNew* and *pNew* itself.

WM_CALCVALIDRECTS - Syntax

This message is sent from [WinSetWindowPos](#) and [WinSetMultWindowPos](#) to determine which areas of a window can be preserved if a window is sized, and which should be redisplayed.

```
param1
    PRECTL  pOldNew /* Window-rectangle structures. */

param2
    PSWP    pNew    /* New window position. */
```

WM_CALCVALIDRECTS - Remarks

This message is *not* sent if this window has the CS_SIZEREDRAW style, indicating size-sensitive window content that must be totally redrawn if sized.

This enables the application to determine if the position of the window has changed as well as its size; this can aid alignment processing.

These rectangles can be modified by the window procedure to cause parts of the window to be redrawn and not preserved.

The window manager tries to preserve the screen image by copying the image described by the old rectangle into the image described by the new rectangle. In this way, an application can control the alignment of the preserved image as well, by changing the origin of the first rectangle.

If no change is made to either rectangle, the entire window area is preserved. If either rectangle is empty, the entire window area is completely redrawn by the operation.

Note: This functionality can be used to optimize window updating when the window is resized. For example, if the application returns that the window is to be aligned with the top-left corner, and the top border is sized, the screen data of the window moves with the top border.

In all instances, the rectangles are intersected with the area of the screen that is actually visible and the valid area of the window. That is, only the window area that contains window information is copied.

For example, consider an application that has two scroll bars, that are children of the client window. When the window is resized, the scroll bars must be completely redrawn. By returning rectangles that exclude the scroll bars, the area of the scroll bars is completely redrawn, thereby preserving only the part of the screen that is worth preserving.

WM_CALCVALIDRECTS - Default Processing

The default window procedure processing is to align the valid area with the top-left of the window by returning:

(CVR_ALIGNTOP | CVR_ALIGNLEFT)

In addition, any child windows intersecting the source rectangle pointed to by *pOldView* of this message, are also offset with the aligned window area.

WM_CALCVALIDRECTS - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

WM_CHAR

WM_CHAR Field - fsflags

fsflags (USHORT)	Keyboard control codes.
KC_CHAR	Indicates that <i>uschr</i> value is valid.
KC_SCANCODE	Indicates that <i>ucscancode</i> is valid. Generally, this is set in all WM_CHAR messages generated from actual operator input. However, if the message has been generated by an application that has issued the WinSetHook function to filter keystrokes, or posted to the application queue, this may not be set.
KC_VIRTUALKEY	Indicates that <i>usvk</i> is valid. Normally <i>usvk</i> should be given precedence when processing the message. Note: For those using hooks, when this bit is set, KC_SCANCODE should usually be set as well.
KC_KEYUP	The event is a key-up transition; otherwise it is a down transition.
KC_PREVDOWN	The key has been previously down; otherwise it has been previously up.
KC_DEADKEY	

The character code is a dead key. The application is responsible for displaying the glyph for the dead key without advancing the cursor.

KC_COMPOSITE

The character code is formed by combining the current key with the previous dead key.

KC_INVALIDCOMP

The character code is not a valid combination with the preceding dead key. The application is responsible for advancing the cursor past the dead-key glyph and then, if the current character is not a space, sounding the alarm and displaying the new character code.

KC_LONEKEY

Indicates if the key is pressed and released without any other keys being pressed or released between the time the key goes down and up.

KC_SHIFT

The SHIFT state is active when key press or release occurred.

KC_ALT

The ALT state is active when key press or release occurred.

KC_CTRL

The CTRL state was active when key press or release occurred.

WM_CHAR Field - ucrepeat

ucrepeat ([UCHAR](#))

Repeat count.

WM_CHAR Field - ucscancode

ucscancode ([UCHAR](#))

Hardware scan code.

A keyboard-generated value that identifies the keyboard event. This is the raw scan code, not the translated scan code.

WM_CHAR Field - usch

usch ([USHORT](#))

Character code.

The character value translation of the keyboard event resulting from the current code page that would apply if the CTRL or ALT keys were not depressed.

WM_CHAR Field - usvk

usvk (USHORT)

Virtual key codes.

A virtual key value translation of the keyboard event resulting from the virtual key code table. The low-order byte contains the **vk** value, and the high-order byte is always set to zero by the standard translate table.

0
This value applies if *fsflags* does not contain KC_VIRTUALKEY.

WM_CHAR Return Value - rc

rc (BOOL)

Processed indicator.

TRUE
Message processed
FALSE
Message ignored.

WM_CHAR - Parameters

fsflags (USHORT)

Keyboard control codes.

KC_CHAR
Indicates that *uschr* value is valid.

KC_SCANCODE
Indicates that *uscancode* is valid.

Generally, this is set in all WM_CHAR messages generated from actual operator input. However, if the message has been generated by an application that has issued the [WinSetHook](#) function to filter keystrokes, or posted to the application queue, this may not be set.

KC_VIRTUALKEY
Indicates that *usvk* is valid.

Normally *usvk* should be given precedence when processing the message.

Note: For those using hooks, when this bit is set, KC_SCANCODE should usually be set as well.

KC_KEYUP
The event is a key-up transition; otherwise it is a down transition.

KC_PREVDOWN
The key has been previously down; otherwise it has been previously up.

KC_DEADKEY
The character code is a dead key. The application is responsible for displaying the glyph for the dead key without advancing the cursor.

KC_COMPOSITE
The character code is formed by combining the current key with the previous dead key.

KC_INVALIDCOMP
The character code is not a valid combination with the preceding dead key. The application is responsible for

advancing the cursor past the dead-key glyph and then, if the current character is not a space, sounding the alarm and displaying the new character code.

KC_LONEKEY	Indicates if the key is pressed and released without any other keys being pressed or released between the time the key goes down and up.
KC_SHIFT	The SHIFT state is active when key press or release occurred.
KC_ALT	The ALT state is active when key press or release occurred.
KC_CTRL	The CTRL state was active when key press or release occurred.

ucrepeat ([UCHAR](#))
Repeat count.

ucscancode ([UCHAR](#))
Hardware scan code.

A keyboard-generated value that identifies the keyboard event. This is the raw scan code, not the translated scan code.

usch ([USHORT](#))
Character code.

The character value translation of the keyboard event resulting from the current code page that would apply if the CTRL or ALT keys were not depressed.

usvk ([USHORT](#))
Virtual key codes.

A virtual key value translation of the keyboard event resulting from the virtual key code table. The low-order byte contains the **vk** value, and the high-order byte is always set to zero by the standard translate table.

0
This value applies if *fsflags* does not contain KC_VIRTUALKEY.

rc ([BOOL](#))
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_CHAR - Syntax

This message is sent when an operator presses a key.

```
param1
    USHORT fsflags    /* Keyboard control codes. */
    UCHAR  ucrepeat   /* Repeat count. */
    UCHAR  ucscancode /* Hardware scan code. */

param2
    USHORT usch       /* Character code. */
    USHORT usvk       /* Virtual key codes. */
```

WM_CHAR - Remarks

This message is posted to the queue associated with the window that has the focus.

The set of keys that causes a WM_CHAR message is device-dependent.

When this message is processed, precedence should normally be given to a valid virtual key if there is one contained in the message.

There are several instances when a window procedure may receive this message with the KC_KEYUP bit set, although it did not receive this message for the down transition of the key.

For example,

- The down transition of the key is translated by the function [WinTranslateAccel](#), into a [WM_COMMAND](#), [WM_SYSCOMMAND](#), [WM_HELP](#), or a [WM_NULL](#) message.
- The key down causes the input focus to change (tab to another window, dismiss a dialog, exit a program, and so on).
- Some other event happens that changes the focus between the time that the key is pressed down and the time that it is released.

Applications should normally only process WM_CHAR messages that do not have the KC_KEYUP bit set.

Except for the special instance where the LONEKEY flag is set on an accelerator key definition, all translations are done on the down stroke of the character.

When the current character is a double-byte character then *param2* contains both bytes of the double-byte character. These bytes are in the order CHAR1FROMMP, CHAR2FROMMP. When the current character is a single-byte character, CHAR2FROMMP contains 0.

WM_CHAR - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message other than to set *rc* to FALSE.

WM_CHAR - Related Messages

Related Messages

- [WM_CHAR \(Default Dialogs\)](#)
- [WM_CHAR \(in Entry Fields\)](#)
- [WM_CHAR \(in Frame Controls\)](#)
- [WM_CHAR \(in List Boxes\)](#)
- [WM_CHAR \(in Multiline Entry Fields\)](#)

WM_CHAR - Examples

This example uses the CHARMMSG macro to process a WM_CHAR message. It first uses the macro to determine if a key was released. It then uses the macro to generate a switch statement based on the character received.

```
MRESULT CALLBACK GenericWndProc(hwnd, usMessage, mp1, mp2)
```

```
HWND    hwnd;  
USHORT  usMessage;
```

```
MPARAM mp1;
MPARAM mp2;
{
    switch (usMessage) {
    case WM_CHAR:
        if (CHARMSG(&usMessage)->fs & KC_KEYUP) {
            switch (CHARMSG(&usMessage)->chr) {
                -----
```

WM_CHAR - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Related Messages](#)
 - [Examples](#)
 - [Glossary](#)
-

WM_CHORD

WM_CHORD Field - fsHitTestres

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

WM_CHORD Return Value - rc

rc ([BOOL](#))
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_CHORD - Parameters

fsHitTestres ([USHORT](#))
Hit-test result.

fsHitTestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see [WM_HITTEST](#).

rc ([BOOL](#))
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_CHORD - Syntax

This message occurs when the operator presses both button one and button two on the pointing device.

```
param2
    USHORT fsHitTestres /* Hit-test result. */
```

WM_CHORD - Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

WM_CHORD - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *rc* to FALSE.

WM_CHORD - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

WM_CLOSE

WM_CLOSE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_CLOSE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_CLOSE Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_CLOSE - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_CLOSE - Syntax

This message is sent to a frame window to indicate that the window is being closed by the user.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

WM_CLOSE - Remarks

This message is sent by the frame to itself as a result of receiving a [WM_SYSCOMMAND](#) message with SC_CLOSE code set. If this message is passed to WinDefDlgProc, this function calls WinDismissDlg and passes the DID_CANCEL result code to it.

WM_CLOSE - Default Processing

The default window procedure posts a [WM_QUIT](#) message to the appropriate queue and sets *ulReserved* to 0.

WM_CLOSE - Related Messages

Related Messages

- [WM_CLOSE \(Default Dialogs\)](#)
- [WM_CLOSE \(in Frame Controls\)](#)

WM_CLOSE - Examples

In this example, the fChanges variable is checked. If it is TRUE, the user is asked if he wants to exit without saving any changes. If the user responds by choosing the No button, zero is returned and the application does not exit. If the user responds by choosing the Yes button, a WM_QUIT message is posted and the application terminates.

```
case WM_CLOSE:
    if (fChanges) {
        if (WinMessageBox(HWND_DESKTOP, hwndClient,
            "Do you want to exit without saving your changes?",
            "", 0, MB_NOICON | MB_YESNO) == MBID_NO)
            return (0L);
    }
    WinPostMsg(hwnd, WM_QUIT, 0L, 0L);
    return (0L);
```

WM_CLOSE - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Related Messages](#)

[Examples](#)

[Glossary](#)

WM_COMMAND

WM_COMMAND Field - uscmd

uscmd ([USHORT](#))

Command value.

It is the responsibility of the application to be able to relate *uscmd* to an application function.

WM_COMMAND Field - ussource

ussource ([USHORT](#))

Source type.

Identifies the type of control:

CMDSRC_PUSHBUTTON

Posted by a push-button control. *uscmd* is the window identity of the push button.

CMDSRC_MENU

Posted by a menu control. *uscmd* is the identity of the menu item.

CMDSRC_ACCELERATOR

Posted as the result of an accelerator. *uscmd* is the accelerator command value.

CMDSRC_FONTDLG

Font dialog. *uscmd* is the identity of the font dialog.

CMDSRC_FILEDLG

File dialog. *uscmd* is the identity of the file dialog.

CMDSRC_OTHER

Other source. *uscmd* gives further control-specific information defined for each control type.

WM_COMMAND Field - uspointer

uspointer (USHORT)
Pointer-device indicator.

TRUE	The message is posted as a result of a pointer-device operation.
FALSE	The message is posted as a result of a keyboard operation.

WM_COMMAND Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_COMMAND - Parameters

uscmd (USHORT)
Command value.

It is the responsibility of the application to be able to relate *uscmd* to an application function.

ussource (USHORT)
Source type.

Identifies the type of control:

CMDSRC_PUSHBUTTON	Posted by a push-button control. <i>uscmd</i> is the window identity of the push button.
CMDSRC_MENU	Posted by a menu control. <i>uscmd</i> is the identity of the menu item.
CMDSRC_ACCELERATOR	Posted as the result of an accelerator. <i>uscmd</i> is the accelerator command value.
CMDSRC_FONTDLG	Font dialog. <i>uscmd</i> is the identity of the font dialog.
CMDSRC_FILEDLG	File dialog. <i>uscmd</i> is the identity of the file dialog.
CMDSRC_OTHER	Other source. <i>uscmd</i> gives further control-specific information defined for each control type.

uspointer (USHORT)
Pointer-device indicator.

TRUE	The message is posted as a result of a pointer-device operation.
FALSE	The message is posted as a result of a keyboard operation.

ulReserved (ULONG)
Reserved value, should be 0.

WM_COMMAND - Syntax

This message occurs when a control has a significant event to notify to its owner, or when a key stroke has been translated by an accelerator table.

```
param1
    USHORT    uscmd        /* Command value. */

param2
    USHORT    ussource     /* Source type. */
    USHORT    uspointer    /* Pointer-device indicator. */
```

WM_COMMAND - Remarks

This message is posted to the queue of the owner of the control.

WM_Command handles popup menu command identifiers for pickup, putdown and cancel drag operations. It determines which items to display based on the state of the lazy drag and droppability of the lazy drag set.

WM_COMMAND - Default Processing

The default window procedure takes no action on this message, other than to set *uiReserved* to 0.

WM_COMMAND - Related Messages

Related Messages

- [WM_COMMAND \(Default Dialogs\)](#)
 - [WM_COMMAND \(in Button Controls\)](#)
 - [WM_COMMAND \(in Menu Controls\)](#)
 - [WM_SYSCOMMAND \(in Title Bar Controls\)](#)
-

WM_COMMAND - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)

WM_CONTEXTMENU

WM_CONTEXTMENU Field - ptspointerpos

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *fPointer* is not set to TRUE.

WM_CONTEXTMENU Field - usReserved

usReserved ([USHORT](#))
Reserved value, 0.

WM_CONTEXTMENU Field - fPointer

fPointer ([USHORT](#))
Input device flag.

TRUE	Message resulted from keyboard event.
FALSE	Message resulted from mouse pointer event.

WM_CONTEXTMENU Return Value - rc

rc ([BOOL](#))
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_CONTEXTMENU - Parameters

- ptspointerpos** (POINTS)
Pointer position.
- The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *fPointer* is not set to TRUE.
- usReserved** (USHORT)
Reserved value, 0.
- fPointer** (USHORT)
Input device flag.
- | | |
|-------|--|
| TRUE | Message resulted from keyboard event. |
| FALSE | Message resulted from mouse pointer event. |
- rc** (BOOL)
Processed indicator.
- | | |
|-------|-------------------|
| TRUE | Message processed |
| FALSE | Message ignored. |

WM_CONTEXTMENU - Syntax

This message occurs when the operator requests a pop-up menu.

```
param1
    POINTS    ptspointerpos    /* Pointer position. */

param2
    USHORT    usReserved       /* Reserved value, 0. */
    USHORT    fPointer         /* Input device flag. */
```

WM_CONTEXTMENU - Remarks

This message is posted to the application queue associated with the window that has the focus, or with the window that is to receive the pointer-button information. This message will result from a mouse event, specified by the system value SV_CONTEXTMENU, or a keyboard event, specified by the system value SV_CONTEXTMENUB.

WM_CONTEXTMENU - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set result to FALSE.

WM_CONTEXTMENU - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_CONTROL

WM_CONTROL Field - id

id ([USHORT](#))
Control-window identity.

This is either the parameter of the [WinCreateWindow](#) function or the identity of an item in a dialog template.

WM_CONTROL Field - usnotifycode

usnotifycode ([USHORT](#))
Notify code.

The meaning of the notify code depends on the type of the control. For details, refer to the section describing that control.

WM_CONTROL Field - ulcontrolspect

ulcontrolspect ([ULONG](#))
Control-specific information.

The meaning of the control-specific information depends on the type of the control. For details, refer to the section describing that control.

WM_CONTROL Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_CONTROL - Parameters

id ([USHORT](#))
Control-window identity.

This is either the parameter of the [WinCreateWindow](#) function or the identity of an item in a dialog template.

usnotifycode ([USHORT](#))
Notify code.

The meaning of the notify code depends on the type of the control. For details, refer to the section describing that control.

ulcontrolspect ([ULONG](#))
Control-specific information.

The meaning of the control-specific information depends on the type of the control. For details, refer to the section describing that control.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_CONTROL - Syntax

This message occurs when a control has a significant event to notify to its owner.

```
param1
    USHORT id           /* Control-window identity. */
    USHORT usnotifycode /* Notify code. */

param2
    ULONG ulcontrolspect /* Control-specific information. */
```

WM_CONTROL - Remarks

This message is sent to the owner of the control, thereby offering it the opportunity to perform some activity before returning to the control.

WM_CONTROL - Default Processing

The default window procedure takes no action on this message, other than to set *uiReserved* to 0.

WM_CONTROL - Related Messages

Related Messages

- [WM_CONTROL](#) (in Button Controls)
 - [WM_CONTROL](#) (in Entry Fields)
 - [WM_CONTROL](#) (Language Support Dialog)
 - [WM_CONTROL](#) (Language Support Window)
 - [WM_CONTROL](#) (in List Boxes)
 - [WM_CONTROL](#) (in Multiline Entry Fields)
 - [WM_CONTROL](#) (in Combination Boxes)
 - [WM_CONTROL](#) (in Spin Button Controls)
-

WM_CONTROL - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_CONTROLPOINTER

WM_CONTROLPOINTER Field - usidCtl

usidCtl ([USHORT](#))
Control identifier.

WM_CONTROLPOINTER Field - hptrNew

hptrNew ([HPOINTER](#))

Handle of the pointing device pointer that the control is to use.

WM_CONTROLPOINTER Return Value - hptrRet

hptrRet ([HPOINTER](#))

Returned pointing device-pointer handle that is then used by the control.

WM_CONTROLPOINTER - Parameters

usidCtl ([USHORT](#))

Control identifier.

hptrNew ([HPOINTER](#))

Handle of the pointing device pointer that the control is to use.

hptrRet ([HPOINTER](#))

Returned pointing device-pointer handle that is then used by the control.

WM_CONTROLPOINTER - Syntax

This message is sent to a owner window of a control when the pointing device pointer moves over the control window, allowing the owner to set the pointing device pointer.

```
param1
    USHORT    usidCtl /* Control identifier. */

param2
    HPOINTER  hptrNew /* Handle of the pointing device pointer that the control is to use. */
```

WM_CONTROLPOINTER - Remarks

The recommended approach for an application, that does not have specific reasons for controlling the pointer appearance, is to pass the message to the default window procedure.

WM_CONTROLPOINTER - Default Processing

The default window procedure returns *hptrNew*.

WM_CONTROLPOINTER - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

WM_CREATE

WM_CREATE Field - ctldata

ctldata ([PVOID](#))

Pointer to control data.

This points to a [Control-Data](#) data structure initialized with the data provided in the parameter of the [WinCreateWindow](#) function. This pointer is also contained in the *pCREATE* parameter.

This parameter MUST be a pointer rather than a long.

The first 2 bytes in the data referenced by this pointer should be the total size of the data referenced by the pointer, (for example, see the [ENTRYFDATA](#) or the [FRAMECDATA](#) structure). PM requires this information to enable it to ensure that the referenced data is accessible to both 16-bit and 32-bit code.

WM_CREATE Field - pCREATE

pCREATE ([PCREATESTRUCT](#))

Create structure.

This points to a [CREATESTRUCT](#) data structure. See the description of *ctldata* for a complete description.

WM_CREATE Return Value - rc

rc (**BOOL**)
Error indicator.

TRUE	Discontinue window creation
FALSE	Continue window creation.

WM_CREATE - Parameters

ctldata (**PVOID**)
Pointer to control data.

This points to a [Control-Data](#) data structure initialized with the data provided in the parameter of the [WinCreateWindow](#) function. This pointer is also contained in the *pCREATE* parameter.

This parameter MUST be a pointer rather than a long.

The first 2 bytes in the data referenced by this pointer should be the total size of the data referenced by the pointer, (for example, see the [ENTRYFDATA](#) or the [FRAMECDATA](#) structure). PM requires this information to enable it to ensure that the referenced data is accessible to both 16-bit and 32-bit code.

pCREATE (**PCREATESTRUCT**)
Create structure.

This points to a [CREATESTRUCT](#) data structure. See the description of *ctldata* for a complete description.

rc (**BOOL**)
Error indicator.

TRUE	Discontinue window creation
FALSE	Continue window creation.

WM_CREATE - Syntax

This message occurs when an application requests the creation of a window.

```
param1  
    PVOID          ctldata /* Pointer to control data. */  
  
param2  
    PCREATESTRUCT pCREATE /* Create structure. */
```

WM_CREATE - Remarks

This message is sent to the window procedure of the window being created, thus offering it an opportunity to initialize that window.

The window procedure receives this after the window is created but before the window becomes visible.

WM_CREATE - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE, which is equivalent to continuing the creation of the window.

WM_CREATE - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_CTLCOLORCHANGE

WM_CTLCOLORCHANGE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_CTLCOLORCHANGE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_CTLCOLORCHANGE Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_CTLCOLORCHANGE - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_CTLCOLORCHANGE - Syntax

This message is sent to all windows when a change is made to the control colors by the `WinSetControlColors` function. The message is sent only if an application or global default color is changed.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

WM_CTLCOLORCHANGE - Remarks

This message is sent by the system when a default control color is changed. System control windows will repaint themselves with the new colors when they receive this message.

WM_CTLCOLORCHANGE - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to the default value of FALSE.

WM_CTLCOLORCHANGE - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)

WM_DESTROY

WM_DESTROY Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_DESTROY Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_DESTROY Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_DESTROY - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_DESTROY - Syntax

This message occurs when an application requests the destruction of a window.

```
param1
    ULONG    ulReserved /* Reserved value, should be 0. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */
```

WM_DESTROY - Remarks

This message is sent to the window procedure of the window being destroyed after it has been hidden on the device, thereby offering it an opportunity to perform some termination action for that window.

WM_DESTROY - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_DESTROY - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_DRAWITEM

WM_DRAWITEM Field - idIdentity

idIdentity ([USHORT](#))
Window identifier.

The window identity of the control sending this notification message.

WM_DRAWITEM Field - ulcontrolspect

ulcontrolspect (ULONG)
Control-specific information.

The meaning of the control-specific information depends on the type of control. For details of each control type, refer to the appropriate section.

WM_DRAWITEM Return Value - rc

rc (BOOL)
Item-drawn indicator.

- | | |
|-------|--|
| TRUE | The owner has drawn the item, and so the control does not draw it. |
| FALSE | If the item contains text and the owner does not draw the item, the owner returns this value and the control draws the item. |
-

WM_DRAWITEM - Parameters

idIdentity (USHORT)
Window identifier.

The window identity of the control sending this notification message.

ulcontrolspect (ULONG)
Control-specific information.

The meaning of the control-specific information depends on the type of control. For details of each control type, refer to the appropriate section.

rc (BOOL)
Item-drawn indicator.

- | | |
|-------|--|
| TRUE | The owner has drawn the item, and so the control does not draw it. |
| FALSE | If the item contains text and the owner does not draw the item, the owner returns this value and the control draws the item. |
-

WM_DRAWITEM - Syntax

This notification is sent to the owner of a control each time an item is to be drawn.

```
param1
    USHORT    idIdentity    /* Window identifier. */

param2
    ULONG     ulcontrolspect /* Control-specific information. */
```

WM_DRAWITEM - Remarks

A control can only display some types of information, and emphasize items in a control-specific manner. Therefore, if special items are to be displayed or emphasized in a special manner, this must be done by the owner window of the control.

The control window procedure generates this message and sends it to the owner of the control, informing the owner that an item is to be drawn, offering the owner the opportunity to draw that item and to indicate that either the item has been drawn or that the control is to draw it.

WM_DRAWITEM - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

WM_DRAWITEM - Related Messages

Related Messages

- [WM_DRAWITEM](#) (in Frame Controls)
 - [WM_DRAWITEM](#) (in List Boxes)
 - [WM_DRAWITEM](#) (in Menu Controls)
-

WM_DRAWITEM - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_ENABLE

WM_ENABLE Field - usnewenabledstate

usnewenabledstate (USHORT)
New enabled state indicator.

TRUE	The window was set to the enabled state.
FALSE	The window was set to the disabled state.

WM_ENABLE Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_ENABLE Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_ENABLE - Parameters

usnewenabledstate (USHORT)
New enabled state indicator.

TRUE	The window was set to the enabled state.
FALSE	The window was set to the disabled state.

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

WM_ENABLE - Syntax

This message notifies a windows of a change to its enable state.

```
param1
    USHORT    usnewenabledstate /* New enabled state indicator. */

param2
    ULONG     ulReserved        /* Reserved value, should be 0. */
```

WM_ENABLE - Remarks

This message is sent to the window procedure of the window whose enable state has been changed, thereby giving it an opportunity to perform some action appropriate to new state of the window.

This is just a notification message. If you want to change the enable state of a window, you would use [WinEnableWindow](#)

WM_ENABLE - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_ENABLE - Related Messages

Related Messages

- [WM_ENABLE](#) (in Button Controls)
- [WM_ENABLE](#) (in Multiline Entry Fields)

WM_ENABLE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_ENDDRAG

WM_ENDDRAG Field - ptspointerpos

ptspointerpos (POINTS)
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *fPointer* is not set to TRUE.

WM_ENDDRAG Field - fPointer

fPointer (USHORT)
Input device flag.

TRUE	Message resulted from pointer event
FALSE	Message resulted from keyboard event.

WM_ENDDRAG Return Value - rc

rc (BOOL)
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_ENDDRAG - Parameters

ptspointerpos (POINTS)
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *fPointer* is not set to TRUE.

fPointer (USHORT)
Input device flag.

TRUE	Message resulted from pointer event
FALSE	Message resulted from keyboard event.

rc (BOOL)

Processed indicator.

TRUE

Message processed

FALSE

Message ignored.

WM_ENDDRAG - Syntax

This message occurs when the operator completes a drag operation.

```
param1
    POINTS  ptspointerpos /* Pointer position. */
param2
    USHORT  fPointer      /* Input device flag. */
```

WM_ENDDRAG - Remarks

This message is posted to the application queue associated with the window that has the focus, or with the window that is to receive the pointer-button information. This message will result from a mouse event, specified by the system value SV_ENDDRAG.

WM_ENDDRAG - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set result to FALSE.

WM_ENDDRAG - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

WM_ENDSELECT

WM_ENDSELECT Field - ptspointerpos

ptspointerpos (POINTS)
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *fPointer* is not set to TRUE.

WM_ENDSELECT Field - fPointer

fPointer (USHORT)
Input device flag.

TRUE	Message resulted from pointer event
FALSE	Message resulted from keyboard event.

WM_ENDSELECT Return Value - rc

rc (BOOL)
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_ENDSELECT - Parameters

ptspointerpos (POINTS)
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *fPointer* is not set to TRUE.

fPointer (USHORT)
Input device flag.

TRUE	Message resulted from pointer event
FALSE	Message resulted from keyboard event.

rc (BOOL)

Processed indicator.

TRUE

Message processed

FALSE

Message ignored.

WM_ENDSELECT - Syntax

This message occurs when the operator either makes a selection or completes a swipe selection.

```
param1
    POINTS ptspointerpos /* Pointer position. */

param2
    USHORT fPointer      /* Input device flag. */
```

WM_ENDSELECT - Remarks

This message is posted to the application queue associated with the window that has the focus, or with the window that is to receive the pointer-button information. This message will result from a mouse event, specified by the system value SV_ENDSELECT.

WM_ENDSELECT - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set result to FALSE.

WM_ENDSELECT - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

WM_ERROR

WM_ERROR Field - userrorcode

userrorcode ([USHORT](#))
Error code.

WM_ERROR Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_ERROR Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_ERROR - Parameters

userrorcode ([USHORT](#))
Error code.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_ERROR - Syntax

This message occurs when an error is detected in a [WinGetMsg](#) or a [WinPeekMsg](#) function.

```
param1
    USHORT    userrorcode    /* Error code. */

param2
    ULONG     ulReserved     /* Reserved value, should be 0. */
```

WM_ERROR - Remarks

The application can detect the error situation after the [WinGetMsg](#) or the [WinPeekMsg](#) function and before the [WinDispatchMsg](#) function.

WM_ERROR - Default Processing

The default window procedure takes no action on this message, other than to set *uiReserved* to 0.

WM_ERROR - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_FOCUSCHANGE

WM_FOCUSCHANGE Field - hwndFocus

hwndFocus ([HWND](#))
Focus window handle.

WM_FOCUSCHANGE Field - usSetFocus

usSetFocus ([USHORT](#))
Focus flag.

TRUE

The window is receiving the focus and *hwndFocus* identifies the window losing the focus.

FALSE

The window is losing the focus and *hwndFocus* identifies the window receiving the focus.

WM_FOCUSCHANGE Field - fsFocusChange

fsFocusChange (USHORT)

Focus changing indicators.

The indicators are passed from the [WinFocusChange](#) function.

WM_FOCUSCHANGE Return Value - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

WM_FOCUSCHANGE - Parameters

hwndFocus (HWND)

Focus window handle.

usSetFocus (USHORT)

Focus flag.

TRUE

The window is receiving the focus and *hwndFocus* identifies the window losing the focus.

FALSE

The window is losing the focus and *hwndFocus* identifies the window receiving the focus.

fsFocusChange (USHORT)

Focus changing indicators.

The indicators are passed from the [WinFocusChange](#) function.

ulReserved (ULONG)

Reserved value, should be 0.

WM_FOCUSCHANGE - Syntax

This message occurs when the window possessing the focus is changed.

param1

```
        HWND    hwndFocus    /* Focus window handle. */  
param2  
    USHORT    usSetFocus    /* Focus flag. */  
    USHORT    fsFocusChange /* Focus changing indicators. */
```

WM_FOCUSCHANGE - Remarks

This message is sent to both the windows gaining and losing the focus.

WM_FOCUSCHANGE - Default Processing

The default window procedure sends this message to the owner or parent, if it exists and is not the desktop. Otherwise, it sets *ulReserved* to 0.

WM_FOCUSCHANGE - Related Messages

Related Messages

- [WM_FOCUSCHANGE \(in Frame Controls\)](#)

WM_FOCUSCHANGE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_FORMATFRAME

WM_FORMATFRAME Field - pswp

pswp ([PSWP](#))
Structure array.

This points to an array that is to hold the [SWP](#) structures.

WM_FORMATFRAME Field - pprectl

pprectl ([PRECTL](#))
Pointer to client window rectangle.

This is typically the window rectangle of *pswp*, but where the window has a wide border, as specified by FCF_DLGBORDER for example, the rectangle is inset by the size of the border.

WM_FORMATFRAME Return Value - ccount

ccount ([USHORT](#))
Count of the number of [SWP](#) arrays returned.

WM_FORMATFRAME - Parameters

pswp ([PSWP](#))
Structure array.

This points to an array that is to hold the [SWP](#) structures.

pprectl ([PRECTL](#))
Pointer to client window rectangle.

This is typically the window rectangle of *pswp*, but where the window has a wide border, as specified by FCF_DLGBORDER for example, the rectangle is inset by the size of the border.

ccount ([USHORT](#))
Count of the number of [SWP](#) arrays returned.

WM_FORMATFRAME - Syntax

This message is sent to a frame window to calculate the sizes and positions of all of the frame controls and the client window.

```
param1  
    PSWP    pswp    /* Structure array. */  
  
param2
```

```
PRECTL pprectl /* Pointer to client window rectangle. */
```

WM_FORMATFRAME - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *ccount* to the default value of 0.

WM_FORMATFRAME - Related Messages

Related Messages

- [WM_FORMATFRAME \(in Frame Controls\)](#)

WM_FORMATFRAME - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_HELP

WM_HELP Field - uscmd

uscmd ([USHORT](#))
Command value.

It is the responsibility of the application to be able to relate *uscmd* to an application function.

WM_HELP Field - ussource

ussource (USHORT)

Source type.

Identifies the type of control:

CMDSRC_PUSHBUTTON

Posted by a push-button control. *uscmd* is the window identity of the push button.

CMDSRC_MENU

Posted by a menu control. *uscmd* is the identity of the menu item.

CMDSRC_ACCELERATOR

Posted as the result of an accelerator. *uscmd* is the accelerator command value.

CMDSRC_OTHER

Other source. *uscmd* gives further control-specific information defined for each control type.

WM_HELP Field - uspointer

uspointer (USHORT)

Pointer-device indicator.

TRUE

If the message is posted as a result of a pointer-device operation

FALSE

If the message is posted as a result of a keyboard operation.

WM_HELP Return Value - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

WM_HELP - Parameters

uscmd (USHORT)

Command value.

It is the responsibility of the application to be able to relate *uscmd* to an application function.

ussource (USHORT)

Source type.

Identifies the type of control:

CMDSRC_PUSHBUTTON

Posted by a push-button control. *uscmd* is the window identity of the push button.

CMDSRC_MENU

Posted by a menu control. *uscmd* is the identity of the menu item.

CMDSRC_ACCELERATOR

Posted as the result of an accelerator. *uscmd* is the accelerator command value.

CMDSRC_OTHER

Other source. *uscmd* gives further control-specific information defined for each control type.

uspointer ([USHORT](#))

Pointer-device indicator.

TRUE

If the message is posted as a result of a pointer-device operation

FALSE

If the message is posted as a result of a keyboard operation.

uiReserved ([ULONG](#))

Reserved value, should be 0.

WM_HELP - Syntax

This message occurs when a control has a significant event to notify to its owner or when a key stroke has been translated by an accelerator table into a WM_HELP.

```
param1
    USHORT  uscmd          /* Command value. */

param2
    USHORT  ussource       /* Source type. */
    USHORT  uspointer      /* Pointer-device indicator. */
```

WM_HELP - Remarks

This message is identical to a [WM_COMMAND](#) message, but implies that the application should respond to this message by displaying help information.

This message is posted to the queue of the owner of the control.

WM_HELP - Default Processing

The default window procedure sends this message to the parent window, if it exists and is not the desktop. Otherwise, it sets *uiReserved* to 0.

WM_HELP - Related Messages

Related Messages

- [WM_HELP \(in Button Controls\)](#)
- [WM_HELP \(in Menu Controls\)](#)

WM_HELP - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Related Messages](#)

[Glossary](#)

WM_HITTEST

WM_HITTEST Field - ptspointerpos

ptspointerpos ([POINTS](#))

Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

WM_HITTEST Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_HITTEST Return Value - ulresult

ulresult ([ULONG](#))

Hit-test indicator.

The application may return one of these values:

HT_NORMAL

The message should be processed as normal. A [WM_MOUSEMOVE](#), [WM_BUTTON2DOWN](#), or

[WM_BUTTON1DOWN](#) message is posted to the window.

HT_TRANSPARENT

The part of the window underneath the pointer is transparent; hit-testing should continue on windows underneath this window, as if the window did not exist.

HT_DISCARD

The message should be discarded; no message is posted to the application.

HT_ERROR

As HT_DISCARD, except that if the message is a button-down message, an alarm sounds and the window concerned is brought to the foreground.

WM_HITTEST - Parameters

ptspointerpos ([POINTS](#))

Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

ulReserved ([ULONG](#))

Reserved value, should be 0.

ulresult ([ULONG](#))

Hit-test indicator.

The application may return one of these values:

HT_NORMAL

The message should be processed as normal. A [WM_MOUSEMOVE](#), [WM_BUTTON2DOWN](#), or [WM_BUTTON1DOWN](#) message is posted to the window.

HT_TRANSPARENT

The part of the window underneath the pointer is transparent; hit-testing should continue on windows underneath this window, as if the window did not exist.

HT_DISCARD

The message should be discarded; no message is posted to the application.

HT_ERROR

As HT_DISCARD, except that if the message is a button-down message, an alarm sounds and the window concerned is brought to the foreground.

WM_HITTEST - Syntax

This message is sent to determine which window is associated with an input from the pointing device.

```
param1
    POINTS   ptspointerpos    /* Pointer position. */

param2
    ULONG    ulReserved       /* Reserved value, should be 0. */
```

WM_HITTEST - Remarks

This message occurs when an application requests a message by issuing a [WinPeekMsg](#) or a [WinGetMsg](#) function.

If the message that is to be retrieved represents a pointer related event, this message is sent to a window to determine whether the message is in fact destined for that window.

This message is only sent if the window class has the CS_HITTEST style set.

Note: The handling of this message determines whether a disabled window can process pointing device events.

WM_HITTEST - Default Processing

The default window procedure takes no action on this message, other than to set *ulresult* to HT_ERROR if the window is disabled, or to HT_NORMAL otherwise.

WM_HITTEST - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_HSCROLL

WM_HSCROLL Field - usidentifier

usidentifier ([USHORT](#))
Scroll bar control window identifier.

WM_HSCROLL Field - sslider

sslider (SHORT)
Slider position.

- 0
Either the operator is not moving the slider with the pointer device, or for the instance where *uscmd* is SB_SLIDERPOSITION the pointer is outside the tracking rectangle when the button is released.
- Other
Slider position.

WM_HSCROLL Field - uscmd

uscmd (USHORT)
Command.

- SB_LINELEFT
Sent if the operator clicks on the left arrow of the scroll bar, or depresses the VK_LEFT key.
- SB_LINERIGHT
Sent if the operator clicks on the right arrow of the scroll bar, or depresses the VK_RIGHT key.
- SB_PAGELEFT
Sent if the operator clicks on the area to the left of the slider, or depresses the VK_PAGELEFT key.
- SB_PAGERIGHT
Sent if the operator clicks on the area to the right of the slider, or depresses the VK_PAGERIGHT key.
- SB_SLIDERPOSITION
Sent to indicate the final position of the slider.
- SB_SLIDERTRACK
If the operator moves the scroll bar slider with the pointer device, this is sent every time the slider position changes.
- SB_ENDSCROLL
Sent when the operator has finished scrolling, but only if the operator has not been doing any absolute slider positioning.

WM_HSCROLL Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_HSCROLL - Parameters

usidentifier (USHORT)
Scroll bar control window identifier.

sslider (SHORT)
Slider position.

0	Either the operator is not moving the slider with the pointer device, or for the instance where <i>uscmd</i> is SB_SLIDERPOSITION the pointer is outside the tracking rectangle when the button is released.
Other	Slider position.
uscmd (USHORT)	Command.
SB_LINELEFT	Sent if the operator clicks on the left arrow of the scroll bar, or depresses the VK_LEFT key.
SB_LINERIGHT	Sent if the operator clicks on the right arrow of the scroll bar, or depresses the VK_RIGHT key.
SB_PAGELEFT	Sent if the operator clicks on the area to the left of the slider, or depresses the VK_PAGELEFT key.
SB_PAGERIGHT	Sent if the operator clicks on the area to the right of the slider, or depresses the VK_PAGERIGHT key.
SB_SLIDERPOSITION	Sent to indicate the final position of the slider.
SB_SLIDERTRACK	If the operator moves the scroll bar slider with the pointer device, this is sent every time the slider position changes.
SB_ENDSCROLL	Sent when the operator has finished scrolling, but only if the operator has not been doing any absolute slider positioning.
ulReserved (ULONG)	Reserved value, should be 0.

WM_HSCROLL - Syntax

This message occurs when a horizontal scroll bar control has a significant event to notify to its owner.

```
param1
    USHORT  usidentifier /* Scroll bar control window identifier. */

param2
    SHORT   sslider      /* Slider position. */
    USHORT  uscmd        /* Command. */
```

WM_HSCROLL - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_HSCROLL - Related Messages

Related Messages

- [WM_HSCROLL \(in Horizontal Scroll Bars\)](#)

WM_HSCROLL - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_INITDLG

WM_INITDLG Field - hwnd

hwnd ([HWND](#))
Focus window handle.

The handle of the control window that is to receive the input focus.

WM_INITDLG Field - pcreate

pcreate ([PVOID](#))
Application-defined data area.

This points to the data area and is passed by the [WinLoadDlg](#), [WinCreateDlg](#), and [WinDlgBox](#) functions in their parameter.

This parameter MUST be a pointer rather than a long.

The first 2 bytes in the data referenced by this pointer should be the total size of the data referenced by the pointer, (for example, see the [ENTRYFDATA](#) or the [FRAMECDATA](#) structure). PM requires this information to enable it to ensure that the referenced data is accessible to both 16-bit and 32-bit code.

WM_INITDLG Return Value - rc

rc (**BOOL**)

Focus set indicator.

TRUE

Focus window is changed. The dialog procedure can change the window to receive the focus, by issuing a [WinSetFocus](#) whose specifies the handle of another control within the dialog box.

FALSE

Focus window is not changed.

WM_INITDLG - Parameters

hwnd (**HWND**)

Focus window handle.

The handle of the control window that is to receive the input focus.

pcreate (**PVOID**)

Application-defined data area.

This points to the data area and is passed by the [WinLoadDlg](#), [WinCreateDlg](#), and [WinDlgBox](#) functions in their parameter.

This parameter MUST be a pointer rather than a long.

The first 2 bytes in the data referenced by this pointer should be the total size of the data referenced by the pointer, (for example, see the [ENTRYFDATA](#) or the [FRAMECDATA](#) structure). PM requires this information to enable it to ensure that the referenced data is accessible to both 16-bit and 32-bit code.

rc (**BOOL**)

Focus set indicator.

TRUE

Focus window is changed. The dialog procedure can change the window to receive the focus, by issuing a [WinSetFocus](#) whose specifies the handle of another control within the dialog box.

FALSE

Focus window is not changed.

WM_INITDLG - Syntax

This message occurs when a dialog box is being created.

```
param1
    HWND    hwnd    /* Focus window handle. */

param2
    PVOID    pcreate /* Application-defined data area. */
```

WM_INITDLG - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_INITDLG - Related Messages

Related Messages

- [WM_INITDLG \(Default Dialogs\)](#)

WM_INITDLG - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_INITMENU

WM_INITMENU Field - smenuid

smenuid ([SHORT](#))
Menu-control identifier.

WM_INITMENU Field - hwnd

hwnd ([HWND](#))
Menu-window handle.

WM_INITMENU Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_INITMENU - Parameters

smenuid ([SHORT](#))
Menu-control identifier.

hwnd ([HWND](#))
Menu-window handle.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_INITMENU - Syntax

This message occurs when a menu control is about to become active.

```
param1  
    SHORT    smenuid    /* Menu-control identifier. */  
  
param2  
    HWND     hwnd       /* Menu-window handle. */
```

WM_INITMENU - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_INITMENU - Related Messages

Related Messages

- [WM_INITMENU](#) (in Frame Controls)
 - [WM_INITMENU](#) (in Menu Controls)
-

WM_INITMENU - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Default Processing](#)

[Related Messages](#)

[Glossary](#)

WM_JOURNALNOTIFY

WM_JOURNALNOTIFY Field - ulCommand

ulCommand ([ULONG](#))

Command to journal.

JRN_QUEUESTATUS

The [WinQueryQueueStatus](#) command must be journaled.

JRN_PHYSKEYSTATE

The [WinGetPhysKeyState](#) command must be journaled.

WM_JOURNALNOTIFY Field - fsQueueStatus

fsQueueStatus ([USHORT](#))

Queue status.

See the parameter of the [WinQueryQueueStatus](#) function.

WM_JOURNALNOTIFY Field - usScanCode

usScanCode ([USHORT](#))

Scan code.

See the parameter of the [WinGetPhysKeyState](#) function.

param2 contains *usScanCode* and *usKeyState* if *ulCommand* has the value JRN_PHYSKEYSTATE.

WM_JOURNALNOTIFY Field - usKeyState

usKeyState (USHORT)
Key State.

See the parameter of the [WinGetPhysKeyState](#) function.

param2 contains *usScanCode* and *usKeyState* if *ulCommand* has the value JRN_PHYSKEYSTATE.

WM_JOURNALNOTIFY Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_JOURNALNOTIFY - Parameters

ulCommand (ULONG)
Command to journal.

JRN_QUEUESTATUS
The [WinQueryQueueStatus](#) command must be journaled.

JRN_PHYSKEYSTATE
The [WinGetPhysKeyState](#) command must be journaled.

fsQueueStatus (USHORT)
Queue status.

See the parameter of the [WinQueryQueueStatus](#) function.

usScanCode (USHORT)
Scan code.

See the parameter of the [WinGetPhysKeyState](#) function.

param2 contains *usScanCode* and *usKeyState* if *ulCommand* has the value JRN_PHYSKEYSTATE.

usKeyState (USHORT)
Key State.

See the parameter of the [WinGetPhysKeyState](#) function.

param2 contains *usScanCode* and *usKeyState* if *ulCommand* has the value JRN_PHYSKEYSTATE.

ulReserved (ULONG)
Reserved value, should be 0.

WM_JOURNALNOTIFY - Syntax

This message is used to maintain correct operation during journal playback.

```
param1
    ULONG    ulCommand    /* Command to journal. */

param2
    USHORT    fsQueueStatus /* Queue status. */
    USHORT    usScanCode   /* Scan code. */
    USHORT    usKeyState   /* Key State. */
```

WM_JOURNALNOTIFY - Remarks

If the [WinQueryQueueStatus](#) or the [WinGetPhysKeyState](#) functions have new information since the last time they were called and there is a journal record hook installed, the journal record hook is called with this message to record this new information.

During playback, this message is interpreted by the system and the appropriate state restored.

Data values of the *param2* parameter depend on which command is to be journaled.

WM_JOURNALNOTIFY - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *ulReserved* to 0.

WM_JOURNALNOTIFY - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_MATCHMNEMONIC

WM_MATCHMNEMONIC Field - usmatch

usmatch ([USHORT](#))
Match character.

WM_MATCHMNEMONIC Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_MATCHMNEMONIC Return Value - rc

rc (BOOL)
Match indicator.

TRUE	Mnemonic found
FALSE	Mnemonic not found, or an error occurred.

WM_MATCHMNEMONIC - Parameters

usmatch (USHORT)
Match character.

ulReserved (ULONG)
Reserved value, should be 0.

rc (BOOL)
Match indicator.

TRUE	Mnemonic found
FALSE	Mnemonic not found, or an error occurred.

WM_MATCHMNEMONIC - Syntax

This message is sent by the dialog box to a control window to determine whether a typed character matches a mnemonic in its window text.

```
param1
    USHORT    usmatch    /* Match character. */

param2
    ULONG     ulReserved /* Reserved value, should be 0. */
```

WM_MATCHMNEMONIC - Default Processing

The default dialog procedure takes no action on this message, other than to set *rc* to FALSE.

WM_MATCHMNEMONIC - Related Messages

Related Messages

- [WM_MATCHMNEMONIC \(in Button Controls\)](#)
 - [WM_MATCHMNEMONIC \(Default Dialogs\)](#)
 - [WM_MATCHMNEMONIC \(in Static Controls\)](#)
-

WM_MATCHMNEMONIC - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Default Processing](#)

[Related Messages](#)

[Glossary](#)

WM_MEASUREITEM

WM_MEASUREITEM Field - sIdentity

sIdentity ([SHORT](#))
Control identifier.

WM_MEASUREITEM Field - ulControlSpec

ulControlSpec ([ULONG](#))
Control-specific information.

The meaning of the control-specific information depends on the type of control. For details of each control type, refer to the appropriate control section.

WM_MEASUREITEM Field - sHeight

sHeight ([SHORT](#))
Height of item.

WM_MEASUREITEM Field - sWidth

sWidth ([SHORT](#))
Width of item.

WM_MEASUREITEM - Parameters

sIdentity ([SHORT](#))
Control identifier.

ulControlSpec ([ULONG](#))
Control-specific information.

The meaning of the control-specific information depends on the type of control. For details of each control type, refer to the appropriate control section.

sHeight ([SHORT](#))
Height of item.

sWidth ([SHORT](#))
Width of item.

WM_MEASUREITEM - Syntax

This notification is sent to the owner of a specific control to establish the height and width for an item in that control.

```
param1
    SHORT sIdentity    /* Control identifier. */

param2
    ULONG ulControlSpec /* Control-specific information. */

returns
    SHORT sHeight      /* Height of item. */
```



```
SHORT sWidth /* Width of item. */
```

WM_MEASUREITEM - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *ReturnCode* to the default value of 0.

WM_MEASUREITEM - Related Messages

Related Messages

- [WM_MEASUREITEM](#) (in Frame Controls)
 - [WM_MEASUREITEM](#) (in List Boxes)
 - [WM_MEASUREITEM](#) (in Menu Controls)
-

WM_MEASUREITEM - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_MENUEND

WM_MENUEND Field - usmenuid

usmenuid ([USHORT](#))
Menu-control identifier.

WM_MENUEND Field - hwnd

hwnd ([HWND](#))
Menu-control window handle.

WM_MENUEND Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_MENUEND - Parameters

usmenuid ([USHORT](#))
Menu-control identifier.

hwnd ([HWND](#))
Menu-control window handle.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_MENUEND - Syntax

This message occurs when a menu control is about to terminate.

```
param1
    USHORT    usmenuid    /* Menu-control identifier. */

param2
    HWND      hwnd        /* Menu-control window handle. */
```

WM_MENUEND - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_MENUEND - Related Messages

Related Messages

- [WM_MENUEND](#) (in [Menu Controls](#))

WM_MENUEND - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Default Processing](#)

[Related Messages](#)

[Glossary](#)

WM_MENUSELECT

WM_MENUSELECT Field - `usItem`

`usItem` ([USHORT](#))

Identifier of selected item.

WM_MENUSELECT Field - `usPostCommand`

`usPostCommand` ([USHORT](#))

Post-command flag.

TRUE

Indicates that either a [WM_COMMAND](#), [WM_SYSCOMMAND](#), or [WM_HELP](#) message is being posted by the menu control on return from the owner, subject to */C*.

FALSE

Indicates that no message is being posted by the menu control on return from the owner, subject to */C*.

WM_MENUSELECT Field - `hwnd`

`hwnd` ([HWND](#))

Menu-control window handle.

WM_MENUSELECT Return Value - rc

rc (BOOL)

Post indicator.

TRUE

Indicates that either a [WM_COMMAND](#), [WM_SYSCOMMAND](#), or [WM_HELP](#) message is to be posted by the menu control window procedure. The menu is dismissed if the selected item does not have a style of [MIA_NODISMISS](#).

FALSE

Indicates that no message is to be posted by the menu control window procedure and that the menu is not dismissed.

WM_MENUSELECT - Parameters

usItem (USHORT)

Identifier of selected item.

usPostCommand (USHORT)

Post-command flag.

TRUE

Indicates that either a [WM_COMMAND](#), [WM_SYSCOMMAND](#), or [WM_HELP](#) message is being posted by the menu control on return from the owner, subject to *rc*.

FALSE

Indicates that no message is being posted by the menu control on return from the owner, subject to *rc*.

hwnd (HWND)

Menu-control window handle.

rc (BOOL)

Post indicator.

TRUE

Indicates that either a [WM_COMMAND](#), [WM_SYSCOMMAND](#), or [WM_HELP](#) message is to be posted by the menu control window procedure. The menu is dismissed if the selected item does not have a style of [MIA_NODISMISS](#).

FALSE

Indicates that no message is to be posted by the menu control window procedure and that the menu is not dismissed.

WM_MENUSELECT - Syntax

This message occurs when a menu item has been selected.

```
param1
    USHORT    usItem        /* Identifier of selected item. */
```

```
        USHORT    usPostCommand    /* Post-command flag. */  
param2  
        HWND      hwnd            /* Menu-control window handle. */
```

WM_MENUSELECT - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to TRUE.

WM_MENUSELECT - Related Messages

Related Messages

- [WM_MENUSELECT \(in Frame Controls\)](#)
 - [WM_MENUSELECT \(in Menu Controls\)](#)
-

WM_MENUSELECT - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_MINMAXFRAME

WM_MINMAXFRAME Field - pswp

pswp ([PSWP](#))

Set window position structure.

This points to a [SWP](#) structure. The structure has the appropriate SWP_* indicators set to describe the operation that is occurring to the window.

WM_MINMAXFRAME Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_MINMAXFRAME Return Value - rc

rc (BOOL)
Processed indicator.

TRUE

The message has been processed; the default system actions for the operation specified by the *pswp* parameter to the window are not to be performed.

FALSE

The message has been ignored; the default system actions for the operation specified by the *pswp* parameter to the window are to be performed.

WM_MINMAXFRAME - Parameters

pswp (PSWP)
Set window position structure.

This points to a **SWP** structure. The structure has the appropriate SWP_* indicators set to describe the operation that is occurring to the window.

ulReserved (ULONG)
Reserved value, should be 0.

rc (BOOL)
Processed indicator.

TRUE

The message has been processed; the default system actions for the operation specified by the *pswp* parameter to the window are not to be performed.

FALSE

The message has been ignored; the default system actions for the operation specified by the *pswp* parameter to the window are to be performed.

WM_MINMAXFRAME - Syntax

This message is sent to a frame window that is being minimized, maximized, or restored.

```
param1
    PSWP    pswp    /* Set window position structure. */
param2
```

```
ULONG ulReserved /* Reserved value, should be 0. */
```

WM_MINMAXFRAME - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_MINMAXFRAME - Related Messages

Related Messages

- [WM_MINMAXFRAME \(in Frame Controls\)](#)

WM_MINMAXFRAME - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_MOUSEMAP

WM_MOUSEMAP Field - ulPhysButton

ulPhysButton ([ULONG](#))

The physical button number (1, 2, or 3).

WM_MOUSEMAP Field - ulMappedButton

ulMappedButton ([ULONG](#))

The button to be mapped to (1, 2, or 3).

WM_MOUSEMAP Return Value - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

WM_MOUSEMAP - Parameters

ulPhysButton (ULONG)

The physical button number (1, 2, or 3).

ulMappedButton (ULONG)

The button to be mapped to (1, 2, or 3).

ulReserved (ULONG)

Reserved value, should be 0.

WM_MOUSEMAP - Syntax

This message is specific to version 2.1, or higher, of the OS/2 operating system.

This message is used only by applications that wish to remap mouse messages in the PM input queue. It is not recommended for general application usage, and applications should NOT process this message in their window procedures.

```
param1
    ULONG   ulPhysButton    /* The physical button number (1, 2, or 3). */

param2
    ULONG   ulMappedButton  /* The button to be mapped to (1, 2, or 3). */
```

WM_MOUSEMAP - Remarks

PM will interpret this message when it is read from the PM input queue, as a request to remap all subsequent mouse events for the desired button, until another WM_MOUSEMAP message is received, cancelling that remap request. This message has no meaning to an application.

WM_MOUSEMAP - Default Processing

The default window procedure takes no action on this message, other than to set *uiReserved* to 0.

WM_MOUSEMAP - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

WM_MOUSEMOVE

WM_MOUSEMOVE Field - sxMouse

sxMouse ([SHORT](#))
Pointing device x-coordinate.

WM_MOUSEMOVE Field - syMouse

syMouse ([SHORT](#))
Pointing device y-coordinate.

WM_MOUSEMOVE Field - uswHitTest

uswHitTest ([USHORT](#))
Message result.

Zero	A pointing device capture is currently in progress
Other	The result of the WM_HITTEST message.

WM_MOUSEMOVE Field - fsflags

fsflags (USHORT)

Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed
---------	----------------------------------

WM_MOUSEMOVE Return Value - rc

rc (BOOL)

Processed indicator.

TRUE	The window procedure did process the message.
FALSE	The window procedure did not process the message.

WM_MOUSEMOVE - Parameters

sxMouse (SHORT)

Pointing device x-coordinate.

syMouse (SHORT)

Pointing device y-coordinate.

uswHitTest (USHORT)

Message result.

Zero	A pointing device capture is currently in progress
Other	The result of the WM_HITTEST message.

fsflags (USHORT)

Keyboard control codes.

In addition to the control codes described with the WM_CHAR message, the following keyboard control codes are valid.

KC_NONE	Indicates that no key is pressed
---------	----------------------------------

rc (BOOL)

Processed indicator.

TRUE	The window procedure did process the message.
FALSE	The window procedure did not process the message.

WM_MOUSEMOVE - Syntax

This message occurs when the pointing device pointer moves.

```
param1
    SHORT    sxMouse    /* Pointing device x-coordinate. */
    SHORT    syMouse    /* Pointing device y-coordinate. */

param2
    USHORT   uswHitTest /* Message result. */
    USHORT   fsflags    /* Keyboard control codes. */
```

WM_MOUSEMOVE - Remarks

The keyboard control codes specified by "flags" reflects the keyboard state at the time the mouse message was initiated. This may or may not reflect the current keyboard state.

param1 contains the position of the pointing device in window coordinates relative to the bottom-left corner of the window.

WM_MOUSEMOVE - Default Processing

The default window procedure sets the pointer shape using the [WinSetPointer](#) function and sets *rc* to FALSE.

WM_MOUSEMOVE - Related Messages

Related Messages

- [WM_MOUSEMOVE](#) (in Multiline Entry Fields)

WM_MOUSEMOVE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_MOVE

WM_MOVE Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_MOVE Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_MOVE Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_MOVE - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

WM_MOVE - Syntax

This message occurs when a window with style CS_MOVENOTIFY changes its absolute position.

```
param1
    ULONG    ulReserved /* Reserved value, should be 0. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */
```

WM_MOVE - Remarks

The message is sent from [WinSetWindowPos](#), [WinSetMultWindowPos](#), and [WinScrollWindow](#).

The message is sent to any window when it is moved relative to its parent window. In addition, a WM_MOVE message is also sent to any children of that window that have style CS_MOVENOTIFY.

The new position of the window is obtained by calling [WinQueryWindowRect](#), and can make those rectangle coordinates relative to any window by calling [WinMapWindowPoints](#).

Note: There are several instances where windows have cause to know if they have been moved, and these include the occasions when the window does not change position relative to its parent, but does change position relative to the screen (its absolute position).

An example is menus. When a top-level menu control (child of the frame window) moves its absolute position as a result of the frame window being moved, the top-level menu control causes the movement of any pull-down menus along with its movement. The same applies to application/dialog box positional grouping. In some instances, a dialog box might cause to be moved as the main window is moved, to make room for other applications.

WM_MOVE - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_MOVE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_MSGBOXDISMISS

WM_MSGBOXDISMISS Field - hwnd

hwnd ([HWND](#))
Non-modal window handle.

WM_MSGBOXDISMISS Field - ulButtonId

ulButtonId ([ULONG](#))
Identity of the selected button in the message box.

WM_MSGBOXDISMISS Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, must be 0.

WM_MSGBOXDISMISS - Parameters

hwnd ([HWND](#))
Non-modal window handle.

ulButtonId ([ULONG](#))
Identity of the selected button in the message box.

ulReserved ([ULONG](#))
Reserved value, must be 0.

WM_MSGBOXDISMISS - Syntax

This message notifies the owner of the message when a non-modal message box has been dismissed (the message box is no longer visible).

```
param1
    HWND    hwnd        /* Non-modal window handle. */

param2
    ULONG    ulButtonId  /* Identity of the selected button in the message box. */
```

WM_MSGBOXDISMISS - Remarks

This message is processed within the owner's window procedure when a non-modal message box is dismissed. It is up to the parent to destroy the message box.

WM_MSGBOXDISMISS - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Glossary](#)

WM_MSGBOXINIT

WM_MSGBOXINIT Field - hwnd

hwnd ([HWND](#))
Non-modal window handle.

WM_MSGBOXINIT Field - idWindow

idWindow ([LONG](#))
Window identity of the message box.

WM_MSGBOXINIT Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, must be 0.

WM_MSGBOXINIT - Parameters

hwnd ([HWND](#))
Non-modal window handle.

idWindow ([LONG](#))
Window identity of the message box.

ulReserved ([ULONG](#))
Reserved value, must be 0.

WM_MSGBOXINIT - Syntax

This message notifies the owner of the message when a non-modal message box has been created and is currently being displayed.

```
param1
    HWND    hwnd           /* Non-modal window handle. */

param2
    LONG    idWindow       /* Window identity of the message box. */
```

WM_MSGBOXINIT - Remarks

This message is processed within the owner's window procedure when a non-modal [WinMessageBox2](#) is created. It is up to the owner to store the window handle returned by this function. This handle is then used to properly destroy the message box when [WM_MSGBOXDISMISS](#) is received or when the parent chooses to destroy it.

WM_MSGBOXINIT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Glossary](#)
-

WM_NEXTMENU

WM_NEXTMENU Field - hwndMenu

hwndMenu ([HWND](#))
Menu-control window handle.

WM_NEXTMENU Field - usPrev

usPrev ([USHORT](#))
Previous-menu indicator.

TRUE	Beginning of the menu has been reached
FALSE	End of the menu has been reached.

WM_NEXTMENU Return Value - hwndNewMenu

hwndNewMenu ([HWND](#))
New menu window handle.

NULLHANDLE	No new menu
Other	New menu window handle.

WM_NEXTMENU - Parameters

hwndMenu ([HWND](#))
Menu-control window handle.

usPrev ([USHORT](#))
Previous-menu indicator.

TRUE	Beginning of the menu has been reached
FALSE	End of the menu has been reached.

hwndNewMenu ([HWND](#))
New menu window handle.

NULLHANDLE	No new menu
Other	New menu window handle.

WM_NEXTMENU - Syntax

This message occurs when either the beginning or the end of the menu is reached by use of the cursor control keys.

```
param1      HWND      hwndMenu      /* Menu-control window handle. */
param2      USHORT     usPrev        /* Previous-menu indicator. */
```

WM_NEXTMENU - Default Processing

The default window procedure takes no action on this message, other than to set *hwndNewMenu* to NULLHANDLE.

WM_NEXTMENU - Related Messages

Related Messages

- [WM_NEXTMENU \(in Frame Controls\)](#)
- [WM_NEXTMENU \(in Menu Controls\)](#)

WM_NEXTMENU - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_NULL

WM_NULL Field - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

WM_NULL Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_NULL Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_NULL - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

WM_NULL - Syntax

This message is posted to activate message queues or modal loops.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

WM_NULL - Remarks

On receiving this message, the application should simply let the default processing take place.

WM_NULL - Default Processing

The default window procedure takes no action on this message, other than to set *uiReserved* to 0.

WM_NULL - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)
-

WM_OPEN

WM_OPEN Field - usPointer

usPointer ([USHORT](#))
Input device flag.

TRUE	Message resulted from pointer event
FALSE	Message resulted from keyboard event

WM_OPEN Field - ptspointerpos

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *usPointer* is not set to TRUE.

WM_OPEN Return Value - rc

rc (**BOOL**)
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_OPEN - Parameters

usPointer (**USHORT**)
Input device flag.

TRUE	Message resulted from pointer event
FALSE	Message resulted from keyboard event

ptspointerpos (**POINTS**)
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *usPointer* is not set to TRUE.

rc (**BOOL**)
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_OPEN - Syntax

This message occurs when the operator makes an *OPEN* request.

```
param1
    USHORT  usPointer      /* Input device flag. */

param2
    POINTS  ptspointerpos /* Pointer position. */
```

WM_OPEN - Remarks

This message is posted to the application queue associated with the window that has the focus, or with the window that is to receive the pointer-button information. This message will result from a mouse event, specified by the system value SV_OPEN.

WM_OPEN - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set result to FALSE.

WM_OPEN - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_PACTIVATE

WM_PACTIVATE Field - usactive

usactive ([USHORT](#))
Active indicator.

TRUE	The window was activated
FALSE	The window was deactivated.

WM_PACTIVATE Field - hwnd

hwnd ([HWND](#))
Window handle.

In the case of activation, *hwnd* identifies the window which was activated. In the case of deactivation, *hwnd* identifies the window which was deactivated.

WM_PACTIVATE Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_PACTIVATE - Parameters

usactive ([USHORT](#))
Active indicator.

TRUE	The window was activated
FALSE	The window was deactivated.

hwnd ([HWND](#))
Window handle.

In the case of activation, *hwnd* identifies the window which was activated. In the case of deactivation, *hwnd* identifies the window which was deactivated.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_PACTIVATE - Syntax

This message is posted when the Language Support Window or Dialog Procedure processes a [WM_ACTIVATE](#) message.

```
param1
    USHORT    usactive    /* Active indicator. */

param2
    HWND      hwnd        /* Window handle. */
```

WM_PACTIVATE - Remarks

The activation change has already occurred when the application receives this message.

WM_PACTIVATE - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_PACTIVATE - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_PAINT

WM_PAINT Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_PAINT Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_PAINT Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_PAINT - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))

Reserved value, should be 0.

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_PAINT - Syntax

This message occurs when a window needs repainting.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

WM_PAINT - Default Processing

The default window procedure issues the [WinBeginPaint](#) and [WinEndPaint](#) functions, and then sets *ulReserved* to 0.

WM_PAINT - Related Messages

Related Messages

- [WM_PAINT](#) (in Frame Controls)
 - [WM_PAINT](#) (Language Support Dialog)
 - [WM_PAINT](#) (Language Support Window)
-

WM_PAINT - Examples

This example shows how an application gets a presentation space for drawing by calling the [WinBeginPaint](#) function. When drawing is complete, the [WinEndPaint](#) function is called to release the presentation space.

```
case WM_PAINT:
    hps = WinBeginPaint(hwnd, NULL, &rc1);
    .
    . /* drawing routines would go here */
    .

    WinEndPaint(hps);
    return (0L);
```

WM_PAINT - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Default Processing
 - Related Messages
 - Examples
 - Glossary

WM_PCONTROL

WM_PCONTROL Field - id

id ([USHORT](#))
Control-window identity.

This is either the parameter of the [WinCreateWindow](#) function or the identity of an item in a dialog template.

WM_PCONTROL Field - usnotifycode

usnotifycode ([USHORT](#))
Notify code.

The meaning of the notify code depends on the type of the control. For details, refer to the section describing that control.

WM_PCONTROL Field - ulZero

ulZero ([ULONG](#))
Zero.

0 The control-specific information in *ulcontrolspec* of the [WM_CONTROL](#) message is not available because the information might not be valid when the application receives this message.

WM_PCONTROL Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_PCONTROL - Parameters

id ([USHORT](#))
Control-window identity.

This is either the parameter of the [WinCreateWindow](#) function or the identity of an item in a dialog template.

usnotifycode ([USHORT](#))
Notify code.

The meaning of the notify code depends on the type of the control. For details, refer to the section describing that control.

ulZero ([ULONG](#))
Zero.

0 The control-specific information in *ulcontrolspec* of the [WM_CONTROL](#) message is not available because the information might not be valid when the application receives this message.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_PCONTROL - Syntax

This message is posted when the Language Support Window or Dialog Procedure processes a [WM_CONTROL](#) message.

```
param1
    USHORT id          /* Control-window identity. */
    USHORT usnotifycode /* Notify code. */

param2
    ULONG ulZero        /* Zero. */
```

WM_PCONTROL - Remarks

The notification from the control has already been processed when the application receives this message.

WM_PCONTROL - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_PCONTROL - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_PPAINT

WM_PPAINT Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_PPAINT Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_PPAINT Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_PPAINT - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_PPAINT - Syntax

This message is posted when the Language Support Window or Dialog Procedure processes a [WM_PAINT](#) message.

```
param1
    ULONG  ulReserved  /* Reserved value, should be 0. */

param2
    ULONG  ulReserved  /* Reserved value, should be 0. */
```

WM_PPAINT - Default Processing

The default window procedure issues the [WinBeginPaint](#) and [WinEndPaint](#) functions, and then sets *ulReserved* to 0.

WM_PPAINT - Related Messages

Related Messages

- [WM_PPAINT \(Language Support Dialog\)](#)
- [WM_PPAINT \(Language Support Window\)](#)

WM_PPAINT - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Default Processing](#)
- [Related Messages](#)
- [Glossary](#)

WM_PRESPARAMCHANGED

WM_PRESPARAMCHANGED Field - idAttrType

idAttrType ([ULONG](#))
Presentation parameter attribute identity.

WM_PRESPARAMCHANGED Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_PRESPARAMCHANGED Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_PRESPARAMCHANGED - Parameters

idAttrType ([ULONG](#))
Presentation parameter attribute identity.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_PRESPARAMCHANGED - Syntax

This message is sent when a presentation parameter is set or removed dynamically from a window instance using the [WinSetPresParam](#) or [WinRemovePresParam](#) functions. It is also sent to all windows owned by the window whose presentation parameter was changed.

```
param1
    ULONG idAttrType /* Presentation parameter attribute identity. */
```

```
param2
    ULONG    ulReserved /* Reserved value, should be 0. */
```

WM_PRESPARAMCHANGED - Remarks

This message notifies a control when an inherited presentation parameter changes.

WM_PRESPARAMCHANGED - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_PRESPARAMCHANGED - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_PSETFOCUS

WM_PSETFOCUS Field - hwnd

hwnd ([HWND](#))

Focus-window handle.

NULLHANDLE

No window lost or received the focus.

Other

Window handle.

WM_PSETFOCUS Field - usfocus

usfocus (USHORT)
Focus flag.

- TRUE
The window received the focus. *hwnd* is the window handle of the window which lost the focus, or NULLHANDLE if no window previously had the focus.
- FALSE
The window lost the focus. *hwnd* is the window handle of the window which received the focus, or NULLHANDLE if no window received the focus.

WM_PSETFOCUS Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_PSETFOCUS - Parameters

hwnd (HWND)
Focus-window handle.

- NULLHANDLE
No window lost or received the focus.
- Other
Window handle.

usfocus (USHORT)
Focus flag.

- TRUE
The window received the focus. *hwnd* is the window handle of the window which lost the focus, or NULLHANDLE if no window previously had the focus.
- FALSE
The window lost the focus. *hwnd* is the window handle of the window which received the focus, or NULLHANDLE if no window received the focus.

ulReserved (ULONG)
Reserved value, should be 0.

WM_PSETFOCUS - Syntax

This message is posted when the Language Support Window or Dialog Procedure processes a [WM_SETFOCUS](#) message.

```
param1  
    HWND    hwnd          /* Focus-window handle. */
```



```
param2
    USHORT    usfocus    /* Focus flag. */
```

WM_PSETFOCUS - Remarks

The focus change has already occurred when the application receives this message.

WM_PSETFOCUS - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_PSETFOCUS - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_PSIZE

WM_PSIZE Field - scxold

scxold ([SHORT](#))
Old horizontal size.

WM_PSIZE Field - scyold

scyold ([SHORT](#))
Old vertical size.

WM_PSIZE Field - scxnew

scxnew ([SHORT](#))
New horizontal size.

WM_PSIZE Field - scynew

scynew ([SHORT](#))
New vertical size.

WM_PSIZE Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_PSIZE - Parameters

scxold ([SHORT](#))
Old horizontal size.

scyold ([SHORT](#))
Old vertical size.

scxnew ([SHORT](#))
New horizontal size.

scynew ([SHORT](#))
New vertical size.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_PSIZE - Syntax

This message is posted when the Language Support Window or Dialog Procedure processes a [WM_SIZE](#) message.

```
param1
    SHORT  scxold    /* Old horizontal size. */
    SHORT  scyold    /* Old vertical size. */

param2
    SHORT  scxnew    /* New horizontal size. */
    SHORT  scynew    /* New vertical size. */
```

WM_PSIZE - Remarks

The size change has already occurred when the application receives this message.

WM_PSIZE - Default Processing

The default window procedure takes no action on this message, other than to set *uiReserved* to 0.

WM_PSIZE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_PSYSCOLORCHANGE

WM_PSYSCOLORCHANGE Field - flOptions

flOptions ([ULONG](#))
Options.

Copied from the parameter of the [WinSetSysColors](#) function.

WM_PSYSCOLORCHANGE Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_PSYSCOLORCHANGE Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_PSYSCOLORCHANGE - Parameters

flOptions (ULONG)
Options.

Copied from the parameter of the [WinSetSysColors](#) function.

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

WM_PSYSCOLORCHANGE - Syntax

This message is posted when the Language Support Window or Dialog Procedure processes a [WM_SYSCOLORCHANGE](#) message.

```
param1
    ULONG flOptions /* Options. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

WM_PSYSCOLORCHANGE - Remarks

All windows in the system are invalidated so that they will be redrawn with the new system color.

WM_PSYSCOLORCHANGE - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_PSYSCOLORCHANGE - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_QUERYACCELTABLE

WM_QUERYACCELTABLE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_QUERYACCELTABLE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_QUERYACCELTABLE Return Value - haccel

- haccel** ([HACCEL](#))
Accelerator table handle.
- NULLHANDLE
No accelerator table is associated with the window.
- Other

The handle of the accelerator table associated with the window.

WM_QUERYACCELTABLE - Parameters

ulReserved ([ULONG](#))

Reserved value, should be 0.

ulReserved ([ULONG](#))

Reserved value, should be 0.

haccel ([HACCEL](#))

Accelerator table handle.

NULLHANDLE

No accelerator table is associated with the window.

Other

The handle of the accelerator table associated with the window.

WM_QUERYACCELTABLE - Syntax

This message returns the handle to the accelerator table of a window.

```
param1
    ULONG    ulReserved /* Reserved value, should be 0. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */
```

WM_QUERYACCELTABLE - Default Processing

The default window procedure takes no action on this message, other than to set *haccel* to NULLHANDLE.

WM_QUERYACCELTABLE - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Default Processing](#)

[Glossary](#)

WM_QUERYCONVERTPOS

WM_QUERYCONVERTPOS Field - pCursorPos

pCursorPos ([PRECTL](#))
Cursor position.

If *usCode* = QCP_CONVERT, *pCursorPos* should be updated to contain the position of the cursor in the window receiving this message. The position is specified as a rectangle in screen coordinates.

If *usCode* = QCP_NOCONVERT, *pCursorPos* should not be updated.

WM_QUERYCONVERTPOS Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_QUERYCONVERTPOS Return Value - usCode

usCode ([USHORT](#))
Conversion code.

QCP_CONVERT

Conversion may be performed for the window with the input focus, *pCursorPos* has been updated to contain the position of the cursor.

QCP_NOCONVERT

Conversion should not be performed, the window with the input focus cannot receive DBCS characters, *pCursorPos* has not been updated.

WM_QUERYCONVERTPOS - Parameters

pCursorPos ([PRECTL](#))
Cursor position.

If *usCode* = QCP_CONVERT, *pCursorPos* should be updated to contain the position of the cursor in the window receiving this message. The position is specified as a rectangle in screen coordinates.

If *usCode* = QCP_NOCONVERT, *pCursorPos* should not be updated.

ulReserved ([ULONG](#))

Reserved value, should be 0.

usCode ([USHORT](#))

Conversion code.

QCP_CONVERT

Conversion may be performed for the window with the input focus, *pCursorPos* has been updated to contain the position of the cursor.

QCP_NOCONVERT

Conversion should not be performed, the window with the input focus cannot receive DBCS characters, *pCursorPos* has not been updated.

WM_QUERYCONVERTPOS - Syntax

This message is sent by an application to determine whether it is appropriate to begin conversion of DBCS characters.

```
param1
    PRECTL  pCursorPos  /* Cursor position. */

param2
    ULONG   ulReserved  /* Reserved value, should be 0. */
```

WM_QUERYCONVERTPOS - Remarks

This message enables a DBCS application to determine whether the window with the input focus can handle DBCS characters. The *pCursorPos* parameter can be used as a guide for positioning any conversion window that the application requires.

WM_QUERYCONVERTPOS - Default Processing

The default window procedure returns QCP_CONVERT, and updates *pCursorPos* to the following values:

- xleft = -1
- ybottom = -1
- xright = 0
- ytop = 0

WM_QUERYCONVERTPOS - Related Messages

Related Messages

- [WM_QUERYCONVERTPOS](#) (in Button Controls)
- [WM_QUERYCONVERTPOS](#) (in Title Bar Controls)

- [WM_QUERYCONVERTPOS](#) (in Entry Fields)
 - [WM_QUERYCONVERTPOS](#) (in Frame Controls)
 - [WM_QUERYCONVERTPOS](#) (in List Boxes)
 - [WM_QUERYCONVERTPOS](#) (in Menu Controls)
 - [WM_QUERYCONVERTPOS](#) (in Scroll Bars)
 - [WM_QUERYCONVERTPOS](#) (in Static Controls)
-

WM_QUERYCONVERTPOS - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Related Messages](#)

[Glossary](#)

WM_QUERYCTLTYPE

WM_QUERYCTLTYPE Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_QUERYCTLTYPE Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_QUERYCTLTYPE Return Value - ulType

ulType ([ULONG](#))

Control type.

CCT_*

The window is a system control window of specified type.

0
The window is not a system control window.

WM_QUERYCTLTYPE - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulType ([ULONG](#))
Control type.

CCT_*
The window is a system control window of specified type.

0
The window is not a system control window.

WM_QUERYCTLTYPE - Syntax

This message is sent by an application to a window to determine the type of control window it is (if any).

```
param1  
    ULONG ulReserved /* Reserved value, should be 0. */  
  
param2  
    ULONG ulReserved /* Reserved value, should be 0. */
```

WM_QUERYCTLTYPE - Remarks

This message can be used to determine whether a window is a system control, and if it is a system control, what type of system control.

WM_QUERYCTLTYPE - Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set *rc* to the default value of FALSE.

WM_QUERYCTLTYPE - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_QUERYHELPINFO

WM_QUERYHELPINFO Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_QUERYHELPINFO Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_QUERYHELPINFO Return Value - lhelpinfo

lhelpinfo ([LONG](#))
Help information.

0	No help information associated with the window.
Other	The help information associated with the window.

WM_QUERYHELPINFO - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

lhelpinfo ([LONG](#))
Help information.

- 0
No help information associated with the window.
- Other
The help information associated with the window.

WM_QUERYHELPINFO - Syntax

This message returns the help instance associated with a frame window.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

WM_QUERYHELPINFO - Default Processing

The default window procedure takes no action on this message, other than to set *lhelpinfo* to 0.

WM_QUERYHELPINFO - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_QUERYTRACKINFO

WM_QUERYTRACKINFO Field - ustflags

ustflags (USHORT)

Tracking flags.

Contains a combination of one or more TF_* flags as defined in the [TRACKINFO](#) structure.

WM_QUERYTRACKINFO Field - ptrackinfo

ptrackinfo (PTRACKINFO)

Track information structure.

This points to a [TRACKINFO](#) structure. The receiver of this message must modify this structure.

WM_QUERYTRACKINFO Return Value - rc

rc (BOOL)

Continue indicator.

TRUE

Continue sizing or moving

FALSE

Terminate sizing or moving.

WM_QUERYTRACKINFO - Parameters

ustflags (USHORT)

Tracking flags.

Contains a combination of one or more TF_* flags as defined in the [TRACKINFO](#) structure.

ptrackinfo (PTRACKINFO)

Track information structure.

This points to a [TRACKINFO](#) structure. The receiver of this message must modify this structure.

rc (BOOL)

Continue indicator.

TRUE

Continue sizing or moving

FALSE

Terminate sizing or moving.

WM_QUERYTRACKINFO - Syntax

The frame control generates this message on receiving a [WM_TRACKFRAME \(in Frame Controls\)](#) message.

```
param1
    USHORT    ustflags    /* Tracking flags. */

param2
    PTRACKINFO ptrackinfo /* Track information structure. */
```

WM_QUERYTRACKINFO - Remarks

This message is sent to the window procedure of the owner of a frame control or title bar control respectively.

The [TRACKINFO](#) data structure specified by the *ptrackinfo* parameter is not initialized before the message is sent. It must be correctly completed before returning.

WM_QUERYTRACKINFO - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_QUERYTRACKINFO - Related Messages

Related Messages

- [WM_TRACKFRAME \(in Title Bar Controls\)](#)

WM_QUERYTRACKINFO - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Related Messages](#)
 - [Glossary](#)

WM_QUERYWINDOWPARAMS

WM_QUERYWINDOWPARAMS Field - pwndparams

pwndparams ([PWNDPARAMS](#))
Window parameter structure.

This points to a window parameter structure; see [WNDPARAMS](#).

The valid values of *fsStatus* are WPM_CCHTEXT, WPM_TEXT, WPM_CBCTLDATA, and WPM_CTLDATA.

The flags in *fsStatus* are cleared as each item is processed. If the call is successful, *fsStatus* is 0. If any item has not been processed, the flag for that item is still set.

WM_QUERYWINDOWPARAMS Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_QUERYWINDOWPARAMS Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WM_QUERYWINDOWPARAMS - Parameters

pwndparams ([PWNDPARAMS](#))
Window parameter structure.

This points to a window parameter structure; see [WNDPARAMS](#).

The valid values of *fsStatus* are WPM_CCHTEXT, WPM_TEXT, WPM_CBCTLDATA, and WPM_CTLDATA.

The flags in *fsStatus* are cleared as each item is processed. If the call is successful, *fsStatus* is 0. If any item has not been processed, the flag for that item is still set.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WM_QUERYWINDOWPARAMS - Syntax

This message occurs when an application queries the window parameters.

```
param1
    PWNDPARAMS  pwndparams /* Window parameter structure. */

param2
    ULONG       ulReserved /* Reserved value, should be 0. */
```

WM_QUERYWINDOWPARAMS - Remarks

If this message is sent to a window of another process, the information in, or identified by, *pwndparams* must be in memory shared by both processes.

WM_QUERYWINDOWPARAMS - Default Processing

The default window procedure sets the *cchText*, *cbPresParams*, and *cbCtlData* parameters of the [WNDPARAMS](#) data structure identified by the *pwndparams* to 0, and sets *rc* to FALSE.

WM_QUERYWINDOWPARAMS - Related Messages

Related Messages

- [WM_QUERYWINDOWPARAMS](#) (in Button Controls)
- [WM_QUERYWINDOWPARAMS](#) (in Entry Fields)
- [WM_QUERYWINDOWPARAMS](#) (in Frame Controls)
- [WM_QUERYWINDOWPARAMS](#) (in List Boxes)
- [WM_QUERYWINDOWPARAMS](#) (in Menu Controls)
- [WM_QUERYWINDOWPARAMS](#) (in Multiline Entry Fields)
- [WM_QUERYWINDOWPARAMS](#) (in Scroll Bars)
- [WM_QUERYWINDOWPARAMS](#) (in Static Controls)
- [WM_QUERYWINDOWPARAMS](#) (in Title Bars)

WM_QUERYWINDOWPARAMS - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_QUIT

WM_QUIT Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_QUIT Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_QUIT Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_QUIT - Parameters

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))
Reserved value, should be 0.

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_QUIT - Syntax

This message is posted to terminate the application.

```
param1
    ULONG   ulReserved /* Reserved value, should be 0. */

param2
    ULONG   ulReserved /* Reserved value, should be 0. */
```

WM_QUIT - Remarks

It causes [WinGetMsg](#) to return as for all other messages.

Note: Applications that call [WinPeekMsg](#) rather than [WinGetMsg](#) should test explicitly for WM_QUIT.

This message should not be dispatched to the default window procedure. The intent of this message is to cause the [WinGetMsg](#) loop to terminate.

Typically this message is posted by the application when the application exit command is selected from the action bar.

This message is also sent to all applications when the system is closing down. To reply to this, the application should either cancel the request by issuing an [WinCancelShutdown](#) function or close itself down by issuing a [WinDestroyMsgQueue](#) function.

WM_QUIT - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_QUIT - Examples

In this example, a WM_CLOSE message is received. If the fChanges flag is set, the application calls a function to determine if the user wants to save the changes before exiting. This function (called QuerySaveFile in this example) asks the user if he wants to save the changes. If the user selects OK, the changes are saved. If the user selects cancel, the function returns this value and the application continues normal execution. Otherwise, it posts a WM_QUIT message to terminate the application.

```
case WM_CLOSE:
    if (fChanges) {
        if (QuerySaveFile(hwnd) == MB_CANCEL) {
            return (0L); /* do not exit after all */
        }
    }
    WinPostMsg(hwnd, WM_QUIT, 0L, 0L);
    return (0L);
```

WM_QUIT - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Examples](#)

[Glossary](#)

WM_REALIZEPALETTE

WM_REALIZEPALETTE Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_REALIZEPALETTE Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_REALIZEPALETTE Return Value - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_REALIZEPALETTE - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

WM_REALIZEPALETTE - Syntax

This message is sent to an application whenever changes have been made to the display hardware physical color table as a result of another application calling [WinRealizePalette](#).

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

WM_REALIZEPALETTE - Remarks

The application should call [WinRealizePalette](#) if it has a palette, or pass it on to the default window procedure if it does not.

If the return value from [WinRealizePalette](#) is greater than 0, the application should invalidate its window to cause a repaint using the newly-realized palette.

WM_REALIZEPALETTE - Default Processing

The default window procedure calls [WinRealizePalette](#) with a NULL parameter. This causes the default palette to be realized. If the return value from [WinRealizePalette](#) is greater than 0, the default window procedure invalidates the window, causing it to be repainted with the newly-realized palette.

WM_REALIZEPALETTE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_SAVEAPPLICATION

WM_SAVEAPPLICATION Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_SAVEAPPLICATION Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_SAVEAPPLICATION Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_SAVEAPPLICATION - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

WM_SAVEAPPLICATION - Syntax

This message is sent by the system to notify an application to save its current state.

```
param1
    ULONG ulReserved /* Reserved value, should be 0. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

WM_SAVEAPPLICATION - Remarks

When an application receives this message, it is expected to save its current state by any convenient method, for example, in a profile or in an auxiliary file.

It is the responsibility of the application to use the saved information, as appropriate, when it is resumed.

Even if the application processes this message, it should also pass it to the default window procedure, by using the [WinDefWindowProc](#) call.

WM_SAVEAPPLICATION - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_SAVEAPPLICATION - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_SEM1

WM_SEM1 Field - flAccumBits

flAccumBits ([ULONG](#))
Semaphore value.

The semaphore values from all the WM_SEM1 messages posted to a queue, are accumulated by a logical-OR operation.

WM_SEM1 Field - ulReserved1

ulReserved1 (ULONG)
Reserved value, should be 0.

WM_SEM1 Return Value - ulReserved2

ulReserved2 (ULONG)
Reserved value, should be 0.

WM_SEM1 - Parameters

flAccumBits (ULONG)
Semaphore value.

The semaphore values from all the WM_SEM1 messages posted to a queue, are accumulated by a logical-OR operation.

ulReserved1 (ULONG)
Reserved value, should be 0.

WM_SEM1 - Syntax

This message is sent or posted by an application.

```
param1
    ULONG flAccumBits /* Semaphore value. */

param2
    ULONG ulReserved1 /* Reserved value, should be 0. */
```

WM_SEM1 - Remarks

If the message is posted, it is merged with any existing WM_SEM1 message on the queue by combining the two *flAccumBits* values using a logical-OR operation.

The WM_SEM1 messages are queued higher than any other type of message.

WM_SEM1 - Default Processing

The default window procedure takes no action on this message, other than to set *uiReserved1* to 0.

WM_SEM1 - Examples

In this example, a thread notifies the client window that it is about to terminate. It sends the constant THREAD3 as the flFlags parameter so that when the client window receives the message, it can tell which thread terminated.

```
#define THREAD1 1      /* bit #1 */
#define THREAD2 2      /* bit #2 */
#define THREAD3 4      /* bit #3 */
VOID FAR Thread3() {
    .
    .
    .
    WinPostMsg(hwndClient, WM_SEM1, (MPARAM) THREAD3, 0);
    DosExit(EXIT_THREAD, 0);
}
```

WM_SEM1 - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Examples](#)
[Glossary](#)

WM_SEM2

WM_SEM2 Field - flAccumBits

flAccumBits ([ULONG](#))
Semaphore value.

The semaphore values from all the WM_SEM2 messages posted to a queue, are accumulated by a logical-OR operation.

WM_SEM2 Field - ulReserved1

ulReserved1 (ULONG)

Reserved value, should be 0.

WM_SEM2 Return Value - ulReserved2

ulReserved2 (ULONG)

Reserved value, should be 0.

WM_SEM2 - Parameters

flAccumBits (ULONG)

Semaphore value.

The semaphore values from all the WM_SEM2 messages posted to a queue, are accumulated by a logical-OR operation.

ulReserved1 (ULONG)

Reserved value, should be 0.

WM_SEM2 - Syntax

This message is sent or posted by an application.

```
param1
    ULONG flAccumBits /* Semaphore value. */

param2
    ULONG ulReserved1 /* Reserved value, should be 0. */
```

WM_SEM2 - Remarks

If the message is posted, it is merged with any existing WM_SEM2 message on the queue by combining the two *flAccumBits* values using a

logical-OR operation.

The WM_SEM2 messages are queued above [WM_SEM3](#) and [WM_SEM4](#) messages, and above any [WM_PAINT](#) or [WM_TIMER](#) messages generated by the system, but lower than any other message.

WM_SEM2 - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved1* to 0.

WM_SEM2 - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_SEM3

WM_SEM3 Field - flAccumBits

flAccumBits ([ULONG](#))
Semaphore value.

The semaphore values from all the WM_SEM3 messages posted to a queue, are accumulated by a logical-OR operation.

WM_SEM3 Field - ulReserved1

ulReserved1 ([ULONG](#))
Reserved value, should be 0.

WM_SEM3 Return Value - ulReserved2

ulReserved2 ([ULONG](#))
Reserved value, should be 0.

WM_SEM3 - Parameters

flAccumBits ([ULONG](#))
Semaphore value.

The semaphore values from all the WM_SEM3 messages posted to a queue, are accumulated by a logical-OR operation.

ulReserved1 ([ULONG](#))
Reserved value, should be 0.

WM_SEM3 - Syntax

This message is sent or posted by an application.

```
param1
    ULONG flAccumBits /* Semaphore value. */

param2
    ULONG ulReserved1 /* Reserved value, should be 0. */
```

WM_SEM3 - Remarks

If the message is posted, it is merged with any existing WM_SEM3 message on the queue by combining the two *flAccumBits* values using a logical-OR operation.

The WM_SEM3 messages are queued above [WM_SEM4](#) messages, and any [WM_TIMER](#) messages generated by the system, but lower than any other message.

WM_SEM3 - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved1* to 0.

WM_SEM3 - Topics

Select an item:

WM_SEM4

WM_SEM4 Field - flAccumBits

flAccumBits ([ULONG](#))
Semaphore value.

The semaphore values from all the WM_SEM4 messages posted to a queue, are accumulated by a logical-OR operation.

WM_SEM4 Field - ulReserved1

ulReserved1 ([ULONG](#))
Reserved value, should be 0.

WM_SEM4 Return Value - ulReserved2

ulReserved2 ([ULONG](#))
Reserved value, should be 0.

WM_SEM4 - Parameters

flAccumBits ([ULONG](#))
Semaphore value.

The semaphore values from all the WM_SEM4 messages posted to a queue, are accumulated by a logical-OR operation.

ulReserved1 ([ULONG](#))
Reserved value, should be 0.

WM_SEM4 - Syntax

This message is sent or posted by an application.

```
param1
    ULONG flAccumBits /* Semaphore value. */

param2
    ULONG ulReserved1 /* Reserved value, should be 0. */
```

WM_SEM4 - Remarks

If the message is posted, it is merged with any existing WM_SEM4 message on the queue by combining the two *flAccumBits* values using a logical-OR operation.

The WM_SEM4 messages are queued lower than any other type of message.

WM_SEM4 - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved1* to 0.

WM_SEM4 - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_SETACCELTABLE

WM_SETACCELTABLE Field - haccelNew

haccelNew ([HACCEL](#))
New accelerator table.

WM_SETACCELTABLE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_SETACCELTABLE Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WM_SETACCELTABLE - Parameters

haccelNew ([HACCEL](#))
New accelerator table.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WM_SETACCELTABLE - Syntax

This message establishes the window accelerator table to be used for translation, when the window is active.

```
param1
    HACCEL haccelNew /* New accelerator table. */
```

```
param2
    ULONG    ulReserved /* Reserved value, should be 0. */
```

WM_SETACCELTABLE - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_SETACCELTABLE - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_SETFOCUS

WM_SETFOCUS Field - hwnd

- hwnd** ([HWND](#))
Focus-window handle.
- | | |
|------------|---|
| NULLHANDLE | No window is losing or receiving the focus. |
| Other | Window handle. |

WM_SETFOCUS Field - usfocus

- usfocus** ([USHORT](#))
Focus flag.
- | | |
|-------|--|
| TRUE | The window is receiving the focus. <i>hwnd</i> is the window handle of the window losing the focus, or NULLHANDLE if no window previously had the focus. |
| FALSE | |

The window is losing the focus. *hwnd* is the window handle of the window receiving the focus, or NULLHANDLE if no window is receiving the focus.

WM_SETFOCUS Return Value - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

WM_SETFOCUS - Parameters

hwnd (HWND)

Focus-window handle.

NULLHANDLE

No window is losing or receiving the focus.

Other

Window handle.

usfocus (USHORT)

Focus flag.

TRUE

The window is receiving the focus. *hwnd* is the window handle of the window losing the focus, or NULLHANDLE if no window previously had the focus.

FALSE

The window is losing the focus. *hwnd* is the window handle of the window receiving the focus, or NULLHANDLE if no window is receiving the focus.

ulReserved (ULONG)

Reserved value, should be 0.

WM_SETFOCUS - Syntax

This message occurs when a window is to receive or lose the input focus.

```
param1
    HWND    hwnd    /* Focus-window handle. */

param2
    USHORT  usfocus /* Focus flag. */
```

WM_SETFOCUS - Remarks

This message is sent to the window receiving or losing the focus, thereby giving it the opportunity to perform some appropriate processing.

Note: Except in the instance of [WM_ACTIVATE](#), with *usactive* set to TRUE, an application processing WM_SETFOCUS or [WM_ACTIVATE](#) messages should not change the focus window or active window. If it does, the focus and active window must be restored before the application returns from processing the message. For this reason, any dialog boxes or windows brought up during the processing of WM_SETFOCUS or [WM_ACTIVATE](#) messages should be system modal.

WM_SETFOCUS - Default Processing

The default window procedure takes no action on this message, other than to set *uiReserved* to 0.

WM_SETFOCUS - Related Messages

Related Messages

- [WM_SETFOCUS \(Language Support Dialog\)](#)
- [WM_SETFOCUS \(Language Support Window\)](#)

WM_SETFOCUS - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_SETHelpInfo

WM_SETHelpInfo Field - lhelpinfo

lhelpinfo ([LONG](#))
New help information.

WM_SETHelpINFO Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_SETHelpINFO Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WM_SETHelpINFO - Parameters

lhelpinfo ([LONG](#))
New help information.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WM_SETHelpINFO - Syntax

This message sets the help instance associated with this frame window when the window is active.

```
param1  
    LONG    lhelpinfo    /* New help information. */  
  
param2  
    ULONG    ulReserved  /* Reserved value, should be 0. */
```

WM_SETHelpInfo - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_SETHelpInfo - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Glossary](#)

WM_SETSELECTION

WM_SETSELECTION Field - usselection

usselection ([USHORT](#))
Selection flag.

TRUE	The window is selected.
FALSE	The window is deselected.

WM_SETSELECTION Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_SETSELECTION Return Value - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_SETSELECTION - Parameters

usselection ([USHORT](#))

Selection flag.

TRUE

The window is selected.

FALSE

The window is deselected.

ulReserved ([ULONG](#))

Reserved value, should be 0.

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_SETSELECTION - Syntax

This message occurs when a window is selected or deselected.

```
param1
    USHORT    usselection    /* Selection flag. */

param2
    ULONG     ulReserved     /* Reserved value, should be 0. */
```

WM_SETSELECTION - Remarks

The window procedure is expected to highlight or unhighlight the selected item of the window, as appropriate.

This message is sent to a window when it loses the focus to another window that it does not own. It allows an application to remove the selection when the focus is removed to another application, but to keep it if, for example, the same application displays a dialog box.

WM_SETSELECTION - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_SETSELECTION - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_SETWINDOWPARAMS

WM_SETWINDOWPARAMS Field - pwndparams

pwndparams ([PWNDPARAMS](#))
Window parameter structure.

This points to a window parameter structure; see [WNDPARAMS](#).

The valid values of *fsStatus* are WPM_TEXT and WPM_CTLDATA.

WM_SETWINDOWPARAMS Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_SETWINDOWPARAMS Return Value - rc

rc ([BOOL](#))
Success indicator.

TRUE	Successful operation
FALSE	Error occurred.

WM_SETWINDOWPARAMS - Parameters

pwndparams ([PWNDPARAMS](#))
Window parameter structure.

This points to a window parameter structure; see [WNDPARAMS](#).

The valid values of *fsStatus* are WPM_TEXT and WPM_CTLDATA.

ulReserved ([ULONG](#))
Reserved value, should be 0.

rc ([BOOL](#))
Success indicator.

TRUE	Successful operation
FALSE	Error occurred.

WM_SETWINDOWPARAMS - Syntax

This message occurs when an application sets or changes the window parameters.

```
param1
    PWNDPARAMS pwndparams /* Window parameter structure. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

WM_SETWINDOWPARAMS - Remarks

If this message is sent to a window of another process, the information in, or identified by, *pwndparams* must be in memory shared by both processes.

WM_SETWINDOWPARAMS - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_SETWINDOWPARAMS - Related Messages

Related Messages

- [WM_SETWINDOWPARAMS](#) (in Button Controls)
- [WM_SETWINDOWPARAMS](#) (in Entry Fields)
- [WM_SETWINDOWPARAMS](#) (in Frame Controls)

- [WM_SETWINDOWPARAMS \(in List Boxes\)](#)
- [WM_SETWINDOWPARAMS \(in Menu Controls\)](#)
- [WM_SETWINDOWPARAMS \(in Multiline Entry Fields\)](#)
- [WM_SETWINDOWPARAMS \(in Scroll Bars\)](#)
- [WM_SETWINDOWPARAMS \(in Static Controls\)](#)
- [WM_SETWINDOWPARAMS \(in Title Bar Controls\)](#)

WM_SETWINDOWPARAMS - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Related Messages](#)

[Glossary](#)

WM_SHOW

WM_SHOW Field - usshow

usshow ([USHORT](#))

Show indicator.

TRUE

Show the window

FALSE

Hide the window.

WM_SHOW Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_SHOW Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_SHOW - Parameters

usshow (USHORT)
Show indicator.

TRUE	Show the window
FALSE	Hide the window.

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

WM_SHOW - Syntax

This message occurs when the WS_VISIBLE state of a window is being changed.

```
param1
    USHORT  usshow      /* Show indicator. */

param2
    ULONG   ulReserved  /* Reserved value, should be 0. */
```

WM_SHOW - Remarks

The message is sent after the visibility state has changed.

In this context, the terms "shown" or "hidden" refer to the state of the WS_VISIBLE style bit. This message is *not* sent when a window is obscured by other windows above it.

WM_SHOW - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_SHOW - Topics

- Select an item:
- Syntax
 - Parameters
 - Returns
 - Remarks
 - Default Processing
 - Glossary

WM_SINGLESELECT

WM_SINGLESELECT Field - ptspointerpos

ptspointerpos ([POINTS](#))
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *usPointer* is not set to TRUE.

WM_SINGLESELECT Field - usPointer

usPointer ([USHORT](#))
Input device flag.

TRUE	Message resulted from pointer event
FALSE	Message resulted from keyboard event.

WM_SINGLESELECT Return Value - rc

rc ([BOOL](#))
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_SINGLESELECT - Parameters

ptspointerpos (**POINTS**)
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *usPointer* is not set to TRUE.

usPointer (**USHORT**)
Input device flag.

TRUE	Message resulted from pointer event
FALSE	Message resulted from keyboard event.

rc (**BOOL**)
Processed indicator.

TRUE	Message processed
FALSE	Message ignored.

WM_SINGLESELECT - Syntax

This message occurs when the operator selects a single object.

```
param1
    POINTS    ptspointerpos    /*  Pointer position. */

param2
    USHORT    usPointer        /*  Input device flag. */
```

WM_SINGLESELECT - Remarks

This message is posted to the application queue associated with the window that has the focus, or with the window that is to receive the pointer-button information. This message will result from a mouse event, specified by the system value SV_SINGLESELECT.

WM_SINGLESELECT - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set *rc* to FALSE.

WM_SINGLESELECT - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_SIZE

WM_SIZE Field - scxold

scxold ([SHORT](#))
Old horizontal size.

WM_SIZE Field - scyold

scyold ([SHORT](#))
Old vertical size.

WM_SIZE Field - scxnew

scxnew ([SHORT](#))
New horizontal size.

WM_SIZE Field - scynew

scynew ([SHORT](#))
New vertical size.

WM_SIZE Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_SIZE - Parameters

scxold ([SHORT](#))
Old horizontal size.

scyold ([SHORT](#))
Old vertical size.

scxnew ([SHORT](#))
New horizontal size.

scynew ([SHORT](#))
New vertical size.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_SIZE - Syntax

This message occurs when a window changes its size.

```
param1
    SHORT  scxold    /* Old horizontal size. */
    SHORT  scyold    /* Old vertical size. */

param2
    SHORT  scxnew    /* New horizontal size. */
    SHORT  scynew    /* New vertical size. */
```

WM_SIZE - Remarks

This message is not sent by [WinCreateWindow](#) when a window is created, and so any size-related processing must be done during the [WM_CREATE](#) message processing in this instance.

This message is sent after the window has been actually sized, but before any repainting has been done. Any resizing or repositioning of child windows that might be necessary as a result of the size change is usually done during the processing of this message.

Note: It is generally unwise to output to the window during the processing of this message, because the area drawn might be redrawn, after the [WM_SIZE](#) processing is complete, by the [WinSetWindowPos](#) function.

The processing of this message for a window which is displaying an advanced VIO presentation space must be carried out by the default advanced VIO window procedure.

WM_SIZE - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_SIZE - Related Messages

Related Messages

- [WM_SIZE \(in Frame Controls\)](#)
 - [WM_SIZE \(Language Support Dialog\)](#)
 - [WM_SIZE \(Language Support Window\)](#)
-

WM_SIZE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_SUBSTITUTESTRING

WM_SUBSTITUTESTRING Field - iindex

iindex ([USHORT](#))
Substitution index.

A value corresponding to the decimal character in the substitution phrase.

WM_SUBSTITUTESTRING Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_SUBSTITUTESTRING Return Value - pString

pString ([PSZ](#))
String to be substituted.

This points to a string (character) buffer.

0
No substitution string
Other
Substitution string.

WM_SUBSTITUTESTRING - Parameters

iindex ([USHORT](#))
Substitution index.

A value corresponding to the decimal character in the substitution phrase.

ulReserved ([ULONG](#))
Reserved value, should be 0.

pString ([PSZ](#))
String to be substituted.

This points to a string (character) buffer.

0
No substitution string
Other
Substitution string.

WM_SUBSTITUTESTRING - Syntax

This message is sent from the [WinSubstituteStrings](#) call.

```
param1  
    USHORT    iindex    /* Substitution index. */  
  
param2  
    ULONG     ulReserved /* Reserved value, should be 0. */
```

WM_SUBSTITUTESTRING - Remarks

The [WinSubstituteStrings](#) call has encountered a substitution phrase in a string. The substitution phrase takes the form "%<digit>", where <digit> is a single decimal character; that is, 0 through 9.

WM_SUBSTITUTESTRING - Default Processing

The default window procedure takes no action on this message, other than to set *pString* to 0.

WM_SUBSTITUTESTRING - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_SYSCOLORCHANGE

WM_SYSCOLORCHANGE Field - flOptions

flOptions ([ULONG](#))
Options.

Copied from the parameter of the [WinSetSysColors](#) function and therefore specifies which palette has been changed.

WM_SYSCOLORCHANGE Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_SYSCOLORCHANGE Return Value - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_SYSCOLORCHANGE - Parameters

flOptions ([ULONG](#))

Options.

Copied from the parameter of the [WinSetSysColors](#) function and therefore specifies which palette has been changed.

ulReserved ([ULONG](#))

Reserved value, should be 0.

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_SYSCOLORCHANGE - Syntax

This message is sent to all main windows when a change is made to the system colors by the [WinSetSysColors](#) function.

```
param1
    ULONG flOptions    /* Options. */

param2
    ULONG ulReserved   /* Reserved value, should be 0. */
```

WM_SYSCOLORCHANGE - Remarks

All windows are invalidated, so that they are redrawn with the new colors. When this message is received, applications that depend on the system colors can query the new color values with the [WinQuerySysColor](#) call.

WM_SYSCOLORCHANGE - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_SYSCOLORCHANGE - Related Messages

Related Messages

- [WM_SYSCOLORCHANGE \(Language Support Dialog\)](#)
- [WM_SYSCOLORCHANGE \(Language Support Window\)](#)

WM_SYSCOLORCHANGE - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_SYSCOMMAND

WM_SYSCOMMAND Field - *uscmd*

uscmd ([USHORT](#))

Command value.

The command value can be one of the SC_* values. It is the responsibility of the application to be able to relate *uscmd* to an application function.

WM_SYSCOMMAND Field - *ussource*

ussource ([USHORT](#))

Source type.

Identifies the type of control:

CMDSRC_PUSHBUTTON

Posted by a push-button control. *uscmd* is the window identifier of the push button.

CMDSRC_MENU

Posted by a menu control. *uscmd* is the identifier of the menu item.

CMDSRC_ACCELERATOR

Posted as the result of an accelerator. *uscmd* is the accelerator command value.

CMDSRC_OTHER

Other source. *uscmd* gives further control-specific information defined for each control type.

WM_SYSCOMMAND Field - uspointer

uspointer (USHORT)

Pointing-device indicator.

TRUE

The message is posted as a result of a pointing-device operation.

FALSE

The message is posted as a result of a keyboard operation.

WM_SYSCOMMAND Return Value - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

WM_SYSCOMMAND - Parameters

uscmd (USHORT)

Command value.

The command value can be one of the SC_* values. It is the responsibility of the application to be able to relate *uscmd* to an application function.

ussource (USHORT)

Source type.

Identifies the type of control:

CMDSRC_PUSHBUTTON

Posted by a push-button control. *uscmd* is the window identifier of the push button.

CMDSRC_MENU

Posted by a menu control. *uscmd* is the identifier of the menu item.

CMDSRC_ACCELERATOR

Posted as the result of an accelerator. *uscmd* is the accelerator command value.

CMDSRC_OTHER

Other source. *uscmd* gives further control-specific information defined for each control type.

uspointer (USHORT)

Pointing-device indicator.

TRUE

The message is posted as a result of a pointing-device operation.

FALSE

The message is posted as a result of a keyboard operation.

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_SYSCOMMAND - Syntax

This message occurs when a control has a significant event to report to its owner or when a key stroke has been translated by an accelerator table.

```
param1
    USHORT  uscmd      /* Command value. */

param2
    USHORT  ussource   /* Source type. */
    USHORT  uspointer  /* Pointing-device indicator. */
```

WM_SYSCOMMAND - Remarks

This message is posted to the queue of the owner of the control, thereby offering it the opportunity to perform some activity as a result.

WM_SYSCOMMAND - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_SYSCOMMAND - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_SYSVALUECHANGED

WM_SYSVALUECHANGED Field - usChangedFirst

usChangedFirst ([USHORT](#))
First system value.

The first of a contiguous set of system values that has been changed.

WM_SYSVALUECHANGED Field - usChangedLast

usChangedLast ([USHORT](#))
Last system value.

The last of a contiguous set of system values that has been changed.

WM_SYSVALUECHANGED Return Value - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_SYSVALUECHANGED - Parameters

usChangedFirst ([USHORT](#))
First system value.

The first of a contiguous set of system values that has been changed.

usChangedLast ([USHORT](#))
Last system value.

The last of a contiguous set of system values that has been changed.

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_SYSVALUECHANGED - Syntax

This message is posted to all main windows when one of the settable system values is changed.

```
param1
    USHORT  usChangedFirst /* First system value. */

param2
    USHORT  usChangedLast /* Last system value. */
```

WM_SYSVALUECHANGED - Remarks

If *usChangedFirst* equals *usChangedLast*, only one system value has changed.

If an application changes the settable system values, it is the responsibility of the application to post this message to all main windows.

This message is processed by WC_FRAME windows by doing any frame-specific processing (such as sending [WM_SETBORDERSIZE](#) messages to the size border if SV_CX/CYSIZEBORDER system values have changed) and then sending the message to the client window if one exists.

This message is only posted when settable system values change.

WM_SYSVALUECHANGED - Default Processing

The default window procedure takes no action on this message, other than to set *uReserved* to 0.

WM_SYSVALUECHANGED - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

WM_TEXTEDIT

WM_TEXTEDIT Field - usPointer

usPointer ([USHORT](#))
Input device flag.

TRUE	Message resulted from pointer event
FALSE	Message resulted from keyboard event.

WM_TEXTEDIT Field - ptspointerpos

ptspointerpos (POINTS)
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *usPointer* is not set to TRUE.

WM_TEXTEDIT Return Value - rc

rc (BOOL)	Processed indicator.
TRUE	Message processed
FALSE	Message ignored.

WM_TEXTEDIT - Parameters

usPointer (USHORT)
Input device flag.

TRUE	Message resulted from pointer event
FALSE	Message resulted from keyboard event.

ptspointerpos (POINTS)
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window. This value is ignored if *usPointer* is not set to TRUE.

rc (BOOL)	Processed indicator.
TRUE	Message processed
FALSE	Message ignored.

WM_TEXTEDIT - Syntax

This message occurs when the operator requests a direct name edit operation.

```
param1
    USHORT  usPointer      /*  Input device flag. */

param2
    POINTS  ptspointerpos  /*  Pointer position. */
```

WM_TEXTEDIT - Remarks

This message is posted to the application queue associated with the window that has the focus, or with the window that is to receive the pointer-button information. This message will result from either a mouse event, specified by the system value SV_TEXTEDIT, or a keyboard event, specified by the system value SV_TEXTEDITKB

WM_TEXTEDIT - Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set result to FALSE.

WM_TEXTEDIT - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Default Processing](#)
 - [Glossary](#)

WM_TIMER

WM_TIMER Field - idTimer

idTimer (USHORT)
Timer identity.

Any timer ids that are not being used must be passed on the default window procedure.

WM_TIMER Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_TIMER Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_TIMER - Parameters

idTimer (USHORT)
Timer identity.

Any timer ids that are not being used must be passed on the default window procedure.

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

WM_TIMER - Syntax

This message is posted when a timer times out.

```
param1
    USHORT idTimer      /* Timer identity. */

param2
    ULONG ulReserved    /* Reserved value, should be 0. */
```


WM_TIMER - Remarks

This message is always queued and is processed specially by the [WinGetMsg](#) and [WinPeekMsg](#) calls, as follows:

1. Timers are processed only by the [WinGetMsg](#) and [WinPeekMsg](#) calls.
2. A timer posts only one WM_TIMER message at a time.
3. WM_TIMER messages are queued lower than all other messages except [WM_SEM4](#) messages.

WM_TIMER - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_TIMER - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

WM_TRACKFRAME

WM_TRACKFRAME Field - fsTrackFlags

fsTrackFlags ([USHORT](#))

Tracking flags.

Contains a combination of one or more TF_* flags; for details, see the [TRACKINFO](#) data structure description.

WM_TRACKFRAME Field - ulReserved

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_TRACKFRAME Return Value - rc

rc ([BOOL](#))

Success indicator

TRUE

The operation is successful.

FALSE

The operation is unsuccessful, or the operation is terminated.

WM_TRACKFRAME - Parameters

fsTrackFlags ([USHORT](#))

Tracking flags.

Contains a combination of one or more TF_* flags; for details, see the [TRACKINFO](#) data structure description.

ulReserved ([ULONG](#))

Reserved value, should be 0.

rc ([BOOL](#))

Success indicator

TRUE

The operation is successful.

FALSE

The operation is unsuccessful, or the operation is terminated.

WM_TRACKFRAME - Syntax

This message is sent to a window whenever it is to be moved or sized.

```
param1
    USHORT fsTrackFlags /* Tracking flags. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

WM_TRACKFRAME - Remarks

Respond to this message by causing a tracking rectangle to be drawn to move or size the window. For information, see [WinTrackRect](#).

WM_TRACKFRAME - Default Processing

None.

WM_TRACKFRAME - Related Messages

Related Messages

- [WM_TRACKFRAME \(in Frame Controls\)](#)
-

WM_TRACKFRAME - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_TRANSLATEACCEL

WM_TRANSLATEACCEL Field - pqmsg

pqmsg ([PQMSG](#))
Pointer to a [QMSG](#) structure.

WM_TRANSLATEACCEL Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_TRANSLATEACCEL Return Value - rc

rc ([BOOL](#))

Translated indicator.

TRUE

The character exists in the accelerator table and has been translated in the [QMSG](#) structure.

FALSE

The character does not exist in the accelerator table or the window does not have an accelerator table.

WM_TRANSLATEACCEL - Parameters

pqmsg ([PQMSG](#))

Pointer to a [QMSG](#) structure.

ulReserved ([ULONG](#))

Reserved value, should be 0.

rc ([BOOL](#))

Translated indicator.

TRUE

The character exists in the accelerator table and has been translated in the [QMSG](#) structure.

FALSE

The character does not exist in the accelerator table or the window does not have an accelerator table.

WM_TRANSLATEACCEL - Syntax

This message is sent to the focus window whenever a [WM_CHAR](#) message occurs.

```
param1
    PQMSG  pqmsg      /* Pointer to a QMSG structure. */

param2
    ULONG  ulReserved  /* Reserved value, should be 0. */
```

WM_TRANSLATEACCEL - Remarks

Normally, this message is not processed by the focus window, but is passed to its parent, which passes it to its parent, until a frame window is reached.

WM_TRANSLATEACCEL - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_TRANSLATEACCEL - Related Messages

Related Messages

- [WM_TRANSLATEACCEL \(in Frame Controls\)](#)

WM_TRANSLATEACCEL - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_TRANSLATEMNEMONIC

WM_TRANSLATEMNEMONIC Field - pqmsg

pqmsg ([PQMSG](#))
Pointer to a QMSG structure. [QMSG](#) structure.

WM_TRANSLATEMNEMONIC Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_TRANSLATEMNEMONIC Return Value - rc

rc (BOOL)

Success indicator.

TRUE

The character has been translated into an accelerator.

FALSE

The character has not been translated into an accelerator.

WM_TRANSLATEMNEMONIC - Parameters

pqmsg (PQMSG)

Pointer to a QMSG structure. QMSG structure.

ulReserved (ULONG)

Reserved value, should be 0.

rc (BOOL)

Success indicator.

TRUE

The character has been translated into an accelerator.

FALSE

The character has not been translated into an accelerator.

WM_TRANSLATEMNEMONIC - Syntax

This message occurs during frame control processing of a WM_TRANSLATEACCEL message.

```
param1
    PQMSG pqmsg    /* Pointer to a QMSG structure. QMSG structure. */
param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

WM_TRANSLATEMNEMONIC - Remarks

This message is sent by the frame control to itself during the processing of a WM_TRANSLATEACCEL message, if the frame control does not translate a character into an accelerator by use of the frame window or queue accelerator tables.

When the frame control receives this message, it sends it to the application menu window, that is the window with identity FID_MENU.

WM_TRANSLATEMNEMONIC - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_TRANSLATEMNEMONIC - Related Messages

Related Messages

- [WM_TRANSLATEMNEMONIC \(in Frame Controls\)](#)

WM_TRANSLATEMNEMONIC - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_UPDATEFRAME

WM_UPDATEFRAME Field - flCreateFlags

flCreateFlags ([ULONG](#))
Frame-creation flags.

Contains the FCF_* flags that indicate which frame controls have been added or removed.

WM_UPDATEFRAME Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_UPDATEFRAME Return Value - rc

rc (BOOL)

Processed indicator.

TRUE

Message processed

FALSE

Message ignored.

WM_UPDATEFRAME - Parameters

flCreateFlags (ULONG)

Frame-creation flags.

Contains the FCF_* flags that indicate which frame controls have been added or removed.

ulReserved (ULONG)

Reserved value, should be 0.

rc (BOOL)

Processed indicator.

TRUE

Message processed

FALSE

Message ignored.

WM_UPDATEFRAME - Syntax

This message is sent by an application after frame controls have been added or removed from the window frame.

```
param1
    ULONG flCreateFlags /* Frame-creation flags. */

param2
    ULONG ulReserved /* Reserved value, should be 0. */
```

WM_UPDATEFRAME - Default Processing

The default window procedure takes no action on this message, other than to set *rc* to FALSE.

WM_UPDATEFRAME - Related Messages

Related Messages

- [WM_UPDATEFRAME](#) (in Frame Controls)

WM_UPDATEFRAME - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_VRNDISABLED

WM_VRNDISABLED Field - mp1

mp1 ([VOID](#))
Reserved value.

WM_VRNDISABLED Field - mp2

mp2 ([VOID](#))
Reserved value.

WM_VRNDISABLED Field - ulReserved

ulReserved ([ULONG](#))
Reserved value, should be 0.

WM_VRNDISABLED - Parameters

mp1 ([VOID](#))

Reserved value.

mp2 ([VOID](#))

Reserved value.

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_VRNDISABLED - Syntax

This message indicates that the window is being sized, or that a [WinLockWindowUpdate](#) has been issued for the window or one of its parent windows. Direct drawing to the window should be suspended.

```
param1
    VOID    mp1        /* Reserved value. */

param2
    VOID    mp2        /* Reserved value. */

returns
    ULONG   ulReserved /* Reserved value, should be 0. */
```

WM_VRNDISABLED - Remarks

The window procedure is expected to suspend direct drawing to the window.

WM_VRNDISABLED - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_VRNDISABLED - Topics

Select an item:

[Syntax](#)

WM_VRNENABLED

WM_VRNENABLED Field - ffVisRgnChanged

ffVisRgnChanged (BOOL)	
Flag indicating whether the visible region has been altered.	
TRUE	The visible region has been altered. The application needs to query the new visible region.
FALSE	The visible region has not been changed.

WM_VRNENABLED Field - mp2

mp2 (VOID)	
Reserved value.	

WM_VRNENABLED Return Value - ulReserved

ulReserved (ULONG)	
Reserved value, should be 0.	

WM_VRNENABLED - Parameters

ffVisRgnChanged (BOOL)	
Flag indicating whether the visible region has been altered.	
TRUE	The visible region has been altered. The application needs to query the new visible region.

FALSE

The visible region has not been changed.

mp2 ([VOID](#))

Reserved value.

ulReserved ([ULONG](#))

Reserved value, should be 0.

WM_VRNENABLED - Syntax

This message tells a window that its visible region is now unlocked and is valid for drawing on. It also contains a message parameter to inform the window if the visible region was changed.

```
param1
    BOOL    ffVisRgnChanged /* Flag indicating whether the visible region has been altered. */

param2
    VOID    mp2           /* Reserved value. */
```

WM_VRNENABLED - Remarks

The visible region, in window coordinates, has been sized, moved or unlocked and drawing can now resume. The *ffVisRgnChanged* parameter is TRUE if the visible region was altered, telling the application whether it needs to recheck the visible area of the window. Direct drawing to the window can be resumed.

WM_VRNENABLED - Default Processing

The default window procedure takes no action on this message, other than to set *ulReserved* to 0.

WM_VRNENABLED - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

WM_VSCROLL

WM_VSCROLL Field - usidentifier

usidentifier (USHORT)
Scroll bar-control window identifier.

WM_VSCROLL Field - sslider

sslider (SHORT)
Slider position.

0	Either the operator is not moving the slider with the pointer device, or for the instance when <i>uscmd</i> is SB_SLIDERPOSITION the pointer is outside the tracking rectangle when the button is released.
Other	Slider position.

WM_VSCROLL Field - uscmd

uscmd (USHORT)
Command.

SB_LINEUP	Sent if the operator clicks on the up arrow of the scroll bar, or presses the VK_UP key.
SB_LINEDOWN	Sent if the operator clicks on the down arrow of the scroll bar, or presses the VK_DOWN key.
SB_PAGEUP	Sent if the operator clicks on the area above the slider, or presses the VK_PAGEUP key.
SB_PAGEDOWN	Sent if the operator clicks on the area below the slider, or presses the VK_PAGEDOWN key.
SB_SLIDERPOSITION	Sent to indicate the final position of the slider.
SB_SLIDERTRACK	If the operator moves the scroll bar slider with the pointer device, this is sent every time the slider position changes.
SB_ENDSCROLL	Sent when the operator has finished scrolling, but only if the operator has not been doing any absolute slider positioning.

WM_VSCROLL Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_VSCROLL - Parameters

usidentifier (USHORT)
Scroll bar-control window identifier.

slider (SHORT)
Slider position.

- 0
Either the operator is not moving the slider with the pointer device, or for the instance when *uscmd* is SB_SLIDERPOSITION the pointer is outside the tracking rectangle when the button is released.
- Other
Slider position.

uscmd (USHORT)
Command.

- SB_LINEUP
Sent if the operator clicks on the up arrow of the scroll bar, or presses the VK_UP key.
- SB_LINEDOWN
Sent if the operator clicks on the down arrow of the scroll bar, or presses the VK_DOWN key.
- SB_PAGEUP
Sent if the operator clicks on the area above the slider, or presses the VK_PAGEUP key.
- SB_PAGEDOWN
Sent if the operator clicks on the area below the slider, or presses the VK_PAGEDOWN key.
- SB_SLIDERPOSITION
Sent to indicate the final position of the slider.
- SB_SLIDERTRACK
If the operator moves the scroll bar slider with the pointer device, this is sent every time the slider position changes.
- SB_ENDSCROLL
Sent when the operator has finished scrolling, but only if the operator has not been doing any absolute slider positioning.

ulReserved (ULONG)
Reserved value, should be 0.

WM_VSCROLL - Syntax

This message occurs when a vertical scroll-bar control has a significant event to notify to its owner.

```
param1
    USHORT  usidentifier /* Scroll bar-control window identifier. */

param2
    SHORT   sslider      /* Slider position. */
    USHORT  uscmd        /* Command. */
```

WM_VSCROLL - Default Processing

The default window procedure takes no action on this message, other than to set *uiReserved* to 0.

WM_VSCROLL - Related Messages

Related Messages

- [WM_VSCROLL \(in Vertical Scroll Bars\)](#)
-

WM_VSCROLL - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Related Messages](#)
[Glossary](#)

WM_WINDOWPOSCHANGED

WM_WINDOWPOSCHANGED Field - pswp

pswp ([PSWP](#))
SWP structures.

This points to two [SWP](#) structures. The first [SWP](#) structure describes the entire new window state, whereas the second structure describes the entire old window state. The *ff* parameter of the first structure contains only those indicators corresponding to the state changes that occurred.

WM_WINDOWPOSCHANGED Field - flAwp

flAwp (ULONG)
Adjust window position status indicators.

The AWF_* flags specify the state change of the frame window.

The return value from the WM_ADJUSTWINDOWPOS message:

0	The SWP_NOADJUST option has been specified.
Other	Adjust window position status indicators.

WM_WINDOWPOSCHANGED Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

WM_WINDOWPOSCHANGED - Parameters

pswp (PSWP)
SWP structures.

This points to two SWP structures. The first SWP structure describes the entire new window state, whereas the second structure describes the entire old window state. The // parameter of the first structure contains only those indicators corresponding to the state changes that occurred.

flAwp (ULONG)
Adjust window position status indicators.

The AWF_* flags specify the state change of the frame window.

The return value from the WM_ADJUSTWINDOWPOS message:

0	The SWP_NOADJUST option has been specified.
Other	Adjust window position status indicators.

ulReserved (ULONG)
Reserved value, should be 0.

WM_WINDOWPOSCHANGED - Syntax

If this message has any of the values of the // parameter of the SWP structure set, with the exception of the SWP_NOADJUST and

SWP_NOREDRA values, it is sent to the window procedure of the window whose position is changed.

This message is also sent if the return value from the [WM_ADJUSTWINDOWPOS](#) is not NULL.

```
param1      PSWP    pswp      /* SWP structures. */
param2      ULONG    flAwp     /* Adjust window position status indicators. */
```

WM_WINDOWPOSCHANGED - Default Processing

The default window procedure sets *uReserved* to 0 and sends the following messages, based on the values of the *flAwp* parameter of the first [SWP](#) data structure:

SWP_SIZE	A WM_SIZE with the new window size from the first SWP structure
SWP_HIDE	A WM_SHOW to hide the new window
SWP_SHOW	A WM_SHOW to show the new window.

WM_WINDOWPOSCHANGED - Examples

This example processes the WM_WINDOWPOSCHANGED message and assigns the two structures to pointers.

```
PSWP pswpNew, pswpOld;

case WM_WINDOWPOSCHANGED:
    pswpNew = PVOIDFROMMP(mp1);
    pswpOld = pswpNew + 1;
```

WM_WINDOWPOSCHANGED - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Default Processing](#)
- [Examples](#)
- [Glossary](#)

Data Types

The following data types are used in Presentation Manager. They are listed in alphabetic order.

ACCEL

Accelerator structure.

```
typedef struct _ACCEL {
    USHORT    fs;    /* Options. */
    USHORT    key;   /* Key. */
    USHORT    cmd;   /* Command code. */
} ACCEL;

typedef ACCEL *PACCEL;
```

ACCEL Field - fs

fs ([USHORT](#))
Options.

ACCEL Field - key

key ([USHORT](#))
Key.

ACCEL Field - cmd

cmd ([USHORT](#))
Command code.

The value to be placed in the *uscmd* parameter of a [WM_HELP](#), a [WM_COMMAND](#), or a [WM_SYSCOMMAND](#).

ACCELTABLE

Accelerator-table structure.

```
typedef struct _ACCELTABLE {
    USHORT      cAccel;      /* Number of accelerator entries. */
    USHORT      codepage;    /* Code page for accelerator entries. */
    ACCEL       aaccel[1];   /* Accelerator entries. */
} ACCELTABLE;

typedef ACCELTABLE *PACCELTABLE;
```

ACCELTABLE Field - cAccel

cAccel (**USHORT**)
Number of accelerator entries.

ACCELTABLE Field - codepage

codepage (**USHORT**)
Code page for accelerator entries.

ACCELTABLE Field - aaccel[1]

aaccel[1] (**ACCEL**)
Accelerator entries.

The default accelerator table has the following 16 entries:

Options		Key	Command
HELP		VIRTUALKEY VK_F1	0
SYSCOMMAND	ALT	VIRTUALKEY VK_F4	SC_CLOSE
SYSCOMMAND	ALT	VIRTUALKEY VK_ENTER	SC_RESTORE
SYSCOMMAND	ALT	VIRTUALKEY VK_NEWLINE	SC_RESTORE
SYSCOMMAND	ALT	VIRTUALKEY VK_F5	SC_RESTORE
SYSCOMMAND	ALT	VIRTUALKEY VK_F6	SC_NEXTFRAME
SYSCOMMAND	ALT	VIRTUALKEY VK_F7	SC_MOVE
SYSCOMMAND	ALT	VIRTUALKEY VK_F8	SC_SIZE
SYSCOMMAND	ALT	VIRTUALKEY VK_F9	SC_MINIMIZE
SYSCOMMAND	ALT	VIRTUALKEY VK_F10	SC_MAXIMIZE
SYSCOMMAND		VIRTUALKEY VK_F10	SC_APPMENU
SYSCOMMAND	LONEKEY	VIRTUALKEY VK_ALT	SC_APPMENU
SYSCOMMAND	LONEKEY	VIRTUALKEY VK_ALTGRAF	SC_APPMENU
SYSCOMMAND	ALT	VIRTUALKEY VK_SPACE	SC_SYSMENU
SYSCOMMAND	SHIFT	VIRTUALKEY VK_ESC	SC_SYSMENU
SYSCOMMAND	CONTROL	VIRTUALKEY VK_ESC	SC_TASKMANAGER

Unsigned integer in the range 0 through 4 294 967 295.

```
typedef unsigned long APIRET;
```

APSZ

An array of pointers to NULL-terminated strings.

```
typedef PSZ APSZ[1];
```

ARCPARAMS

Arc-parameters structure.

```
typedef struct _ARCPARAMS {
    LONG    lP; /* P coefficient. */
    LONG    lQ; /* Q coefficient. */
    LONG    lR; /* R coefficient. */
    LONG    lS; /* S coefficient. */
} ARCPARAMS;

typedef ARCPARAMS *PARCPARAMS;
```

ARCPARAMS Field - IP

IP (LONG)
P coefficient.

ARCPARAMS Field - IQ

IQ (LONG)
Q coefficient.

ARCPARAMS Field - IR

IR (**LONG**)
R coefficient.

ARCPARAMS Field - IS

IS (**LONG**)
S coefficient.

AREABUNDLE

Area-attributes bundle structure.

```
typedef struct _AREABUNDLE {  
    LONG      lColor;           /* Area foreground color. */  
    LONG      lBackColor;      /* Area background color. */  
    USHORT    usMixMode;       /* Area foreground-mix mode. */  
    USHORT    usBackMixMode;   /* Area background-mix mode. */  
    USHORT    usSet;           /* Pattern set. */  
    USHORT    usSymbol;        /* Pattern symbol. */  
    POINTL    ptlRefPoint;     /* Pattern reference point. */  
} AREABUNDLE;  
  
typedef AREABUNDLE *PAREABUNDLE;
```

AREABUNDLE Field - IColor

IColor (**LONG**)
Area foreground color.

AREABUNDLE Field - IBackColor

IBackColor (**LONG**)
Area background color.

AREABUNDLE Field - usMixMode

usMixMode (**USHORT**)

Area foreground-mix mode.

AREABUNDLE Field - usBackMixMode

usBackMixMode (USHORT)
Area background-mix mode.

AREABUNDLE Field - usSet

usSet (USHORT)
Pattern set.

AREABUNDLE Field - usSymbol

usSymbol (USHORT)
Pattern symbol.

AREABUNDLE Field - ptlRefPoint

ptlRefPoint (POINTL)
Pattern reference point.

ATOM

Atom identity.
`typedef ULONG ATOM;`

BITMAPARRAYFILEHEADER

Bitmap array file header structure.

```
typedef struct _BITMAPARRAYFILEHEADER {
    USHORT      usType;      /* Type of structure. */
    ULONG       cbSize;      /* Size of the BITMAPARRAYFILEHEADER structure in bytes. */
    ULONG       offNext;     /* Offset of the next BITMAPARRAYFILEHEADER structure from the start of the file. */
    USHORT      cxDisplay;   /* Device width, in pels. */
    USHORT      cyDisplay;   /* Device height, in pels. */
    BITMAPFILEHEADER bfh;    /* Bitmap file header structure. */
} BITMAPARRAYFILEHEADER;

typedef BITMAPARRAYFILEHEADER *PBITMAPARRAYFILEHEADER;
```

BITMAPARRAYFILEHEADER Field - usType

usType (**USHORT**)
Type of structure.

Possible values are shown in the following list:

BFT_BITMAPARRAY
(0x4142 - 'BA' for BITMAPARRAYFILEHEADER or BITMAPARRAYFILEHEADER2)

BITMAPARRAYFILEHEADER Field - cbSize

cbSize (**ULONG**)
Size of the BITMAPARRAYFILEHEADER structure in bytes.

BITMAPARRAYFILEHEADER Field - offNext

offNext (**ULONG**)
Offset of the next BITMAPARRAYFILEHEADER structure from the start of the file.

BITMAPARRAYFILEHEADER Field - cxDisplay

cxDisplay (**USHORT**)
Device width, in pels.

BITMAPARRAYFILEHEADER Field - cyDisplay

cyDisplay ([USHORT](#))
Device height, in pels.

BITMAPARRAYFILEHEADER Field - bfh

bfh ([BITMAPFILEHEADER](#))
Bitmap file header structure.

BITMAPARRAYFILEHEADER2

Bitmap array file header structure.

```
typedef struct _BITMAPARRAYFILEHEADER2 {
    USHORT      usType;      /* Type of structure. */
    ULONG       cbSize;      /* Size of the BITMAPARRAYFILEHEADER2 structure in bytes. */
    ULONG       offNext;     /* Offset of the next BITMAPARRAYFILEHEADER2 structure from the start of t
    USHORT      cxDisplay;    /* Device width, in pels. */
    USHORT      cyDisplay;    /* Device height, in pels. */
    BITMAPFILEHEADER2 bfh2;   /* Bitmap file header structure. */
} BITMAPARRAYFILEHEADER2;

typedef BITMAPARRAYFILEHEADER2 *PBITMAPARRAYFILEHEADER2;
```

BITMAPARRAYFILEHEADER2 Field - usType

usType ([USHORT](#))
Type of structure.

Possible values are shown in the following list:

```
BFT_BITMAPARRAY
(0x4142. = 'BA' for BITMAPARRAYFILEHEADER or BITMAPARRAYFILEHEADER2)
```

BITMAPARRAYFILEHEADER2 Field - cbSize

cbSize ([ULONG](#))
Size of the BITMAPARRAYFILEHEADER2 structure in bytes.

BITMAPARRAYFILEHEADER2 Field - offNext

offNext ([ULONG](#))

Offset of the next BITMAPARRAYFILEHEADER2 structure from the start of the file.

BITMAPARRAYFILEHEADER2 Field - cxDisplay

cxDisplay ([USHORT](#))

Device width, in pels.

BITMAPARRAYFILEHEADER2 Field - cyDisplay

cyDisplay ([USHORT](#))

Device height, in pels.

BITMAPARRAYFILEHEADER2 Field - bfh2

bfh2 ([BITMAPFILEHEADER2](#))

Bitmap file header structure.

BITMAPFILEHEADER

Bitmap file header structure.

```
typedef struct _BITMAPFILEHEADER {
    USHORT      usType;      /* Type of resource the file contains. */
    ULONG       cbSize;      /* Size of the BITMAPFILEHEADER structure in bytes. */
    SHORT       xHotspot;    /* Width of hotspot for icons and pointers. */
    SHORT       yHotspot;    /* Height of hotspot for icons and pointers. */
    USHORT      offBits;     /* Offset in bytes. */
    BITMAPINFOHEADER bmp;    /* Bitmap information header structure. */
} BITMAPFILEHEADER;

typedef BITMAPFILEHEADER *PBITMAPFILEHEADER;
```

BITMAPFILEHEADER Field - usType

usType (USHORT)

Type of resource the file contains.

Possible values are shown in the following list:

- BFT_BMAP (0x4D42 - 'BM' for bitmaps)
- BFT_ICON (0x4349 - 'IC' for icons)
- BFT_POINTER (0x4540 - 'PT' for pointers)
- BFT_COLORICON (0x4943 - 'CI' for color icons)
- BFT_COLORPOINTER (0x5043 - 'CP' for color pointers)

BITMAPFILEHEADER Field - cbSize

cbSize (ULONG)

Size of the BITMAPFILEHEADER structure in bytes.

BITMAPFILEHEADER Field - xHotspot

xHotspot (SHORT)

Width of hotspot for icons and pointers.

This field is ignored for bit maps.

BITMAPFILEHEADER Field - yHotspot

yHotspot (SHORT)

Height of hotspot for icons and pointers.

This field is ignored for bit maps.

BITMAPFILEHEADER Field - offBits

offBits (USHORT)

Offset in bytes.

Offset in bytes to beginning of bitmap pel data in the file, from the start of the definition.

BITMAPFILEHEADER Field - bmp

bmp ([BITMAPINFOHEADER](#))
Bitmap information header structure.

BITMAPFILEHEADER2

Bitmap file header structure.

```
typedef struct _BITMAPFILEHEADER2 {  
    USHORT      usType;      /* Type of resource the file contains. */  
    ULONG       cbSize;      /* Size of the BITMAPFILEHEADER2 structure in bytes. */  
    SHORT       xHotspot;    /* Width of hotspot for icons and pointers. */  
    SHORT       yHotspot;    /* Height of hotspot for icons and pointers. */  
    USHORT      offBits;     /* Offset in bytes. */  
    BITMAPINFOHEADER2 bmp2;   /* Bitmap information header structure. */  
} BITMAPFILEHEADER2;  
  
typedef BITMAPFILEHEADER2 *PBITMAPFILEHEADER2;
```

BITMAPFILEHEADER2 Field - usType

usType ([USHORT](#))
Type of resource the file contains.

Possible values are shown in the following list:

BFT_BMAP	(0x4D42 - 'BM' for bitmaps)
BFT_ICON	(0x4349 - 'IC' for icons)
BFT_POINTER	(0x4540 - 'PT' for pointers)
BFT_COLORICON	(0x4943 - 'CI' for color icons)
BFT_COLORPOINTER	(0x5043 - 'CP' for color pointers)

BITMAPFILEHEADER2 Field - cbSize

cbSize ([ULONG](#))
Size of the BITMAPFILEHEADER2 structure in bytes.

BITMAPFILEHEADER2 Field - xHotspot

xHotspot ([SHORT](#))

Width of hotspot for icons and pointers.

This field is ignored for bit maps.

BITMAPFILEHEADER2 Field - yHotspot

yHotspot ([SHORT](#))

Height of hotspot for icons and pointers.

This field is ignored for bit maps.

BITMAPFILEHEADER2 Field - offBits

offBits ([USHORT](#))

Offset in bytes.

Offset in bytes to beginning of bitmap pel data in the file, from the start of the definition.

BITMAPFILEHEADER2 Field - bmp2

bmp2 ([BITMAPINFOHEADER2](#))

Bitmap information header structure.

BITMAPINFO

Bitmap information structure.

Each bit plane logically contains (*cx* * *cy* * *cBitCount*) bits, although the actual length can be greater because of padding.

See also [BITMAPINFO2](#), which is preferred.

```
typedef struct _BITMAPINFO {  
    ULONG      cbFix;          /* Length of fixed portion of structure. */  
    USHORT     cx;             /* Bitmap width in pels. */  
    USHORT     cy;             /* Bitmap height in pels. */  
    USHORT     cPlanes;        /* Number of bit planes. */  
};
```

```
    USHORT    cBitCount;    /* Number of bits per pel within a plane. */
    RGB        argbColor[1]; /* Array of RGB values. */
} BITMAPINFO;

typedef BITMAPINFO *PBITMAPINFO;
```

BITMAPINFO Field - cbFix

cbFix ([ULONG](#))
Length of fixed portion of structure.

This length can be determined using `sizeof(BITMAPINFOHEADER)`.

BITMAPINFO Field - cx

cx ([USHORT](#))
Bitmap width in pels.

BITMAPINFO Field - cy

cy ([USHORT](#))
Bitmap height in pels.

BITMAPINFO Field - cPlanes

cPlanes ([USHORT](#))
Number of bit planes.

BITMAPINFO Field - cBitCount

cBitCount ([USHORT](#))
Number of bits per pel within a plane.

BITMAPINFO Field - argbColor[1]

argbColor[1] (RGB)

Array of RGB values.

This is a packed array of 24-bit RGB values. If there are N bits per pel ($N = cPlanes * cBitCount$), the array contains $2*N$ RGB values. However, if $N = 24$, the bit map does not need the **color** *color* array because the standard-format bit map, with 24 bits per pel, is assumed to contain RGB values.

BITMAPINFO2

Bitmap information structure.

Each bit plane logically contains ($cx * cy * cBitCount$) bits, although the actual length can be greater because of padding.

Note: Many functions can accept either this structure or the [BITMAPINFO](#) structure. Where possible, BITMAPINFO2 should be used.

The *cbFix* field is used to find the color table, if any, that goes with the information in this structure. A color table is an array of color ([RGB2](#)) values. If there are N bits per pel ($N = cPlanes * cBitCount$), the array contains $2*N$ color values. However, if $N = 24$, the color table is not required (because the standard-format bit map, with 24 bits per pel, is assumed to contain RGB values), unless either *cclrUsed* or *cclrImportant* is non-zero.

```
typedef struct _BITMAPINFO2 {
    ULONG      cbFix;           /* Length of fixed portion of structure. */
    ULONG      cx;             /* Bitmap width in pels. */
    ULONG      cy;             /* Bitmap height in pels. */
    USHORT     cPlanes;        /* Number of bit planes. */
    USHORT     cBitCount;       /* Number of bits per pel within a plane. */
    ULONG      ulCompression;   /* Compression scheme used to store the bit map. */
    ULONG      cbImage;         /* Length of bitmap storage data, in bytes. */
    ULONG      cxResolution;    /* Horizontal component of the resolution of target device. */
    ULONG      cyResolution;    /* Vertical component of the resolution of the target device. */
    ULONG      cclrUsed;        /* Number of color indexes used. */
    ULONG      cclrImportant;    /* Minimum number of color indexes for satisfactory appearance of the bit map. */
    USHORT     usUnits;         /* Units of measure. */
    USHORT     usReserved;      /* Reserved. */
    USHORT     usRecording;     /* Recording algorithm. */
    USHORT     usRendering;     /* Halftoning algorithm. */
    ULONG      cSize1;          /* Size value 1. */
    ULONG      cSize2;          /* Size value 2. */
    ULONG      ulColorEncoding; /* Color encoding. */
    ULONG      ulIdentifier;    /* Reserved for application use. */
    RGB2       argbColor[1];    /* Array of RGB values. */
} BITMAPINFO2;

typedef BITMAPINFO2 *PBITMAPINFO2;
```

BITMAPINFO2 Field - cbFix

cbFix (ULONG)

Length of fixed portion of structure.

The structure can be truncated after *cBitCount* or any subsequent field.

The length does not include the length of the color table. Where the color table is present, it is at an offset of *cbFix* from the start of the BITMAPINFO2 structure.

This length can range from 16 ([BITMAPINFOHEADER](#) through field *cBitCount*) up to sizeof(BITMAPINFOHEADER2) bytes.

BITMAPINFO2 Field - cx

cx ([ULONG](#))
Bitmap width in pels.

BITMAPINFO2 Field - cy

cy ([ULONG](#))
Bitmap height in pels.

BITMAPINFO2 Field - cPlanes

cPlanes ([USHORT](#))
Number of bit planes.

BITMAPINFO2 Field - cBitCount

cBitCount ([USHORT](#))
Number of bits per pel within a plane.

BITMAPINFO2 Field - ulCompression

ulCompression ([ULONG](#))
Compression scheme used to store the bit map.

BCA_UNCOMP
Bit map is uncompressed.

BCA_HUFFMAN1D
The bit map is compressed by a modified Huffman encoding. This is valid for a bi-level (one bit per pel) bit map.

BCA_RLE4

The bit map is a 4-bit per pel run-length encoded bit map. See the following section, "Format of Compressed Data," for a description of the format of the compressed data.

BCA_RLE8

The bit map is an 8-bit per pel run-length encoded bit map. See the following section, "Format of Compressed Data," for a description of the format of the compressed data.

BCA_RLE24

The bit map is a 24-bit per pel run-length encoded bit map. See the following section, "Format of Compressed Data," for a description of the format of the compressed data.

Format of Compressed Data

Encoding a run length:

Run-length encoded bit maps are encoded in the buffer in a controlled format. In all cases, if the first byte is non-zero, it is the length of a run of pels of a particular color or, in the case of a BCA_RLE4 bit map, a run of a length of pels of alternating colors.

1st-byte	pel repetition count ≥ 1
2nd-4th bytes	(BCA_RLE24 only) RGB value of pel.
2nd-byte	(BCA_RLE8) color index of pel to be repeated
	(BCA_RLE4) the second byte contains 2 4-bit color indexes. The repetition count is completed by alternately choosing the high-order nibble followed by the low-order nibble for the succeeding pels until the count is exhausted.

Unencoded run:

An unencoded run is a string of pels to be placed in consecutive positions in the destination bit map.

1st-byte	0
2nd-byte	COUNT = a multiple of 3 for BCA_RLE24 bit maps, or COUNT ≥ 3 (for BCA_RLE4 and BCA_RLE8 bit maps).

followed by the bytes as follows:

BCA_RLE24

A string of bytes specifying the RGB color values of succeeding pels. If COUNT is odd, it must be padded by a zero byte for an even length overall.

BCA_RLE8

A string of bytes specifying color indexes for succeeding pels. If COUNT is odd, it must be padded by a zero byte for an even length overall.

BCA_RLE4

A string of bytes, each byte providing two color indexes, with the high-order nibble specifying the index of the pel preceding the low-order nibble. The COUNT specifies the number of indexes. The overall length of the string must be an even number of bytes, and thus may be padded with a zero byte, and the low order nibble of the last significant byte may also be zero and not used.

Delta record:

A delta record indicates a shift in position in the destination bit map before the next record is decoded.

1st-byte	0
2nd-byte	2
3rd-byte	Delta-x (unsigned)
4th-byte	Delta-y (unsigned)

This is a relative jump record. It implies that the next record is to be decoded into a position in the destination bit map at an offset from the current position, determined by changing the horizontal and vertical positions by Delta-x and Delta-y, respectively.

End-of-line record:

The end-of-line record signifies that the data for the current scan line is complete and that decoding of the next record should begin at the start of the next scan line.

1st-byte	0
2nd-byte	0

End-of-RLE record:

The end-of-RLE record signifies the end of the data in the RLE compressed bit map.

1st-byte	0
2nd-byte	1

BITMAPINFO2 Field - cblImage

cblImage (ULONG)
Length of bitmap storage data, in bytes.

If the bit map is uncompressed, zero (default) can be specified for this.

BITMAPINFO2 Field - cxResolution

cxResolution (ULONG)
Horizontal component of the resolution of target device.

The resolution of the device the bit map is intended for, in the units specified by *usUnits*. This information enables an application to select from a resource group the bit map that best matches the characteristics of the current output device.

BITMAPINFO2 Field - cyResolution

cyResolution (ULONG)
Vertical component of the resolution of the target device.

See the description of *cxResolution*.

BITMAPINFO2 Field - cclrUsed

cclrUsed (ULONG)

Number of color indexes used.

The number of color indexes from the color table that are used by the bit map. If it is zero (the default), all the indexes are used. If it is non-zero, only the first *colorUsed* entries in the table are accessed by the system, and further entries can be omitted.

For the standard formats with a *cBitCount* of 1, 4, or 8 (and *cPlanes* equal to 1), any indexes beyond *ccrUsed* are not valid. For example, a bit map with 64 colors can use the 8-bitcount format without having to supply the other 192 entries in the color table. For the 24-bitcount standard format, *ccrUsed* is the number of colors used by the bit map.

BITMAPINFO2 Field - cclrlImportant

cclrlImportant (ULONG)

Minimum number of color indexes for satisfactory appearance of the bit map.

More colors may be used in the bit map, but it is not necessary to assign them to the device palette. These additional colors may be mapped to the nearest colors available.

Zero (the default) means that all entries are important.

For a 24-bitcount standard format bit map, the *colorImportant* colors are also listed in the color table following the BITMAPINFO2 structure.

BITMAPINFO2 Field - usUnits

usUnits (USHORT)

Units of measure.

Units of measure of the horizontal and vertical components of resolution, *cxResolution* and *cyResolution*.

BRU_METRIC

Pels per meter. This is the default value.

BITMAPINFO2 Field - usReserved

usReserved (USHORT)

Reserved.

This is a reserved field.

BITMAPINFO2 Field - usRecording

usRecording (USHORT)

Recording algorithm.

The format in which the bit map data is recorded.

BRA_BOTTOMUP

Scan lines are recorded bottom to top. This is the default value.

BITMAPINFO2 Field - usRendering

usRendering (USHORT)

Halftoning algorithm.

The algorithm used to record bit map data that has been digitally halftoned.

BRH_NOTHALFTONED

Bitmap data is not halftoned. This is the default value.

BRH_ERRORDIFFUSION

Error Diffusion or Damped Error Diffusion algorithm.

BRH_PANDA

Processing Algorithm for Non-coded Document Acquisition.

BRH_SUPERCIRCLE

Super Circle algorithm.

BITMAPINFO2 Field - cSize1

cSize1 (ULONG)

Size value 1.

If BRH_ERRORDIFFUSION is specified in *usRendering*, *cSize1* is the error damping as a percentage in the range 0 through 100. A value of 100% indicates no damping, and a value of 0% indicates that any errors are not diffused.

If BRH_PANDA or BRH_SUPERCIRCLE is specified, *cSize1* is the x dimension of the pattern used, in pels.

BITMAPINFO2 Field - cSize2

cSize2 (ULONG)

Size value 2.

If BRH_ERRORDIFFUSION is specified in *usRendering*, this parameter is ignored.

If BRH_PANDA or BRH_SUPERCIRCLE is specified, *cSize2* is the y dimension of the pattern used, in pels.

BITMAPINFO2 Field - ulColorEncoding

ulColorEncoding (ULONG)

Color encoding.

BCE_RGB

Each element in the color array is an [RGB2](#) datatype. This is the default value.

BITMAPINFO2 Field - ulIdentifier

ulIdentifier ([ULONG](#))

Reserved for application use.

BITMAPINFO2 Field - argbColor[1]

argbColor[1] ([RGB2](#))

Array of RGB values.

This is a packed array of 24-bit RGB values. If there are N bits per pel ($N = cPlanes * cBitCount$), the array contains 2^{**N} RGB values. However, if $N = 24$, the bit map does not need the **color** array because the standard-format bit map, with 24 bits per pel, is assumed to contain RGB values.

BITMAPINFOHEADER

Bitmap information header structure.

Each bit plane logically contains ($cx * cy * cBitCount$) bits, although the actual length can be greater because of padding.

See also BITMAPINFOHEADER2, which is preferred.

```
typedef struct _BITMAPINFOHEADER {
    ULONG      cbFix;          /* Length of structure. */
    USHORT     cx;             /* Bitmap width in pels. */
    USHORT     cy;             /* Bitmap height in pels. */
    USHORT     cPlanes;        /* Number of bit planes. */
    USHORT     cBitCount;      /* Number of bits per pel within a plane. */
} BITMAPINFOHEADER;

typedef BITMAPINFOHEADER *PBITMAPINFOHEADER;
```

BITMAPINFOHEADER Field - cbFix

cbFix ([ULONG](#))

Length of structure.

BITMAPINFOHEADER Field - cx

cx ([USHORT](#))
Bitmap width in pels.

BITMAPINFOHEADER Field - cy

cy ([USHORT](#))
Bitmap height in pels.

BITMAPINFOHEADER Field - cPlanes

cPlanes ([USHORT](#))
Number of bit planes.

BITMAPINFOHEADER Field - cBitCount

cBitCount ([USHORT](#))
Number of bits per pel within a plane.

BITMAPINFOHEADER2

Bitmap information header structure.

Each bit plane logically contains ($cx * cy * cBitCount$) bits, although the actual length can be greater because of padding.

Note: Many functions can accept either this structure or the [BITMAPINFOHEADER](#) structure. Where possible, use BITMAPINFOHEADER2.

```
typedef struct _BITMAPINFOHEADER2 {
    ULONG      cbFix;           /* Length of structure. */
    ULONG      cx;             /* Bitmap width in pels. */
    ULONG      cy;             /* Bitmap height in pels. */
    USHORT     cPlanes;        /* Number of bit planes. */
    USHORT     cBitCount;       /* Number of bits per pel within a plane. */
    ULONG      ulCompression;   /* Compression scheme used to store the bit map. */
    ULONG      cbImage;         /* Length of bitmap storage data, in bytes. */
    ULONG      cxResolution;    /* Horizontal component of the resolution of target device. */
    ULONG      cyResolution;    /* Vertical component of the resolution of target device. */
    ULONG      cClrUsed;        /* Number of color indexes used. */
    ULONG      cClrImportant;   /* Minimum number of color indexes for satisfactory appearance of the bit map. */
    USHORT     usUnits;         /* Units of measure. */
    USHORT     usReserved;      /* Reserved. */
    USHORT     usRecording;     /* Recording algorithm. */
    USHORT     usRendering;     /* Halftoning algorithm. */
    ULONG      cSize1;          /* Size value 1. */
}
```

```
    ULONG      cSize2;           /* Size value 2. */
    ULONG      ulColorEncoding;  /* Color encoding. */
    ULONG      ulIdentifier;     /* Reserved for application use. */
} BITMAPINFOHEADER2;

typedef BITMAPINFOHEADER2 *PBITMAPINFOHEADER2;
```

BITMAPINFOHEADER2 Field - cbFix

cbFix (ULONG)
Length of structure.

The structure can be truncated after *cBitCount* or any subsequent field.

BITMAPINFOHEADER2 Field - cx

cx (ULONG)
Bitmap width in pels.

BITMAPINFOHEADER2 Field - cy

cy (ULONG)
Bitmap height in pels.

BITMAPINFOHEADER2 Field - cPlanes

cPlanes (USHORT)
Number of bit planes.

BITMAPINFOHEADER2 Field - cBitCount

cBitCount (USHORT)
Number of bits per pel within a plane.

BITMAPINFOHEADER2 Field - ulCompression

ulCompression (ULONG)

Compression scheme used to store the bit map.

BCA_UNCOMP

Bit map is uncompressed.

BCA_HUFFMAN1D

The bit map is compressed by a modified Huffman encoding. This is valid for a bi-level (one bit per pel) bit map.

BCA_RLE4

The bit map is a 4-bit per pel run-length encoded bit map. See the following section, "Format of Compressed Data," for a description of the format of the compressed data.

BCA_RLE8

The bit map is an 8-bit per pel run-length encoded bit map. See the following section, "Format of Compressed Data," for a description of the format of the compressed data.

BCA_RLE24

The bit map is a 24-bit per pel run-length encoded bit map. See the following section, "Format of Compressed Data," for a description of the format of the compressed data.

Format of Compressed Data

Encoding a run length:

Run length encoded bit maps are encoded in the buffer in a controlled format. In all cases, if the first byte is non-zero, it is the length of a run of pels of a particular color or, in the case of a BCA_RLE4 bit map, a run of a length of pels of alternating colors.

1st-byte pel repetition count ≥ 1
2nd-4th bytes (BCA_RLE24 only) RGB value of pel.
2nd-byte (BCA_RLE8) color index of pel to be repeated
 (BCA_RLE4) the second byte contains 2 4-bit
 color indexes. The repetition count is
 completed by alternately choosing the high-order
 nibble followed by the low-order nibble for the
 succeeding pels until the count is exhausted.

Unencoded run:

An unencoded run is a string of pels to be placed in consecutive positions in the destination bit map.

1st-byte 0
2nd-byte COUNT = a multiple of 3 for BCA_RLE24 bit maps, or
 COUNT ≥ 3 (for BCA_RLE4 and BCA_RLE8 bit maps).

followed by the bytes as follows:

BCA_RLE24

A string of bytes specifying the RGB color values of succeeding pels. If COUNT is odd, it must be padded by a zero byte for an even length overall.

BCA_RLE8

A string of bytes specifying color indexes for succeeding pels. If COUNT is odd, it must be padded by a zero byte for an even length overall.

BCA_RLE4

A string of bytes, each byte providing two color indexes, with the high-order nibble specifying the index of the

pel preceding the low-order nibble. The COUNT specifies the number of indexes. The overall length of the string must be an even number of bytes, and thus may be padded with a zero byte, and the low order nibble of the last significant byte may also be zero and not used.

Delta record:

A delta record indicates a shift in position in the destination bit map before the next record is decoded.

1st-byte	0
2nd-byte	2
3rd-byte	Delta-x (unsigned)
4th-byte	Delta-y (unsigned)

This is a relative jump record. It implies that the next record is to be decoded into a position in the destination bit map at an offset from the current position, determined by changing the horizontal and vertical positions by Delta-x and Delta-y, respectively.

End-of-line record:

The end-of-line record signifies that the data for the current scan line is complete and that decoding of the next record should begin at the start of the next scan line.

1st-byte	0
2nd-byte	0

End-of-RLE record:

The end-of-RLE record signifies the end of the data in the RLE compressed bit map.

1st-byte	0
2nd-byte	1

BITMAPINFOHEADER2 Field - cbImage

cbImage (ULONG)

Length of bitmap storage data, in bytes.

If the bit map is uncompressed, zero (the default) can be specified for this.

BITMAPINFOHEADER2 Field - cxResolution

cxResolution (ULONG)

Horizontal component of the resolution of target device.

The resolution of the device the bit map is intended for, in the units specified by *usUnits*. This information enables applications to select from a resource group the bit map that best matches the characteristics of the current output device.

BITMAPINFOHEADER2 Field - cyResolution

cyResolution (ULONG)
Vertical component of the resolution of target device.

See the description of *cxResolution*.

BITMAPINFOHEADER2 Field - cclrUsed

cclrUsed (ULONG)
Number of color indexes used.

The number of color indexes from the color table that are used by the bit map. If this is zero (the default), all the indexes are used. If it is non-zero, only the first *cclrUsed* entries in the table are accessed by the system, and further entries can be omitted.

For the standard formats with a *cBitCount* of 1, 4, or 8 (and *cPlanes* equal to 1), any indexes beyond *cclrUsed* are invalid. For example, a bit map with 64 colors can use the 8-bitcount format without having to supply the other 192 entries in the color table. For the 24-bitcount standard format, *cclrUsed* is the number of colors used by the bit map.

BITMAPINFOHEADER2 Field - cclrImportant

cclrImportant (ULONG)
Minimum number of color indexes for satisfactory appearance of the bit map.

More colors may be used in the bit map, but it is not necessary to assign them to the device palette. These additional colors may be mapped to the nearest colors available.

Zero (the default) means that all entries are important.

For a 24-bitcount standard format bit map, the *cclrImportant* colors are also listed in the color table relating to this bit map.

BITMAPINFOHEADER2 Field - usUnits

usUnits (USHORT)
Units of measure.

Units of measure of the horizontal and vertical resolution, *cxResolution* and *cyResolution*.

BRU_METRIC	Pels per meter. This is the default value.
------------	--

BITMAPINFOHEADER2 Field - usReserved

usReserved (USHORT)

Reserved.

This is a reserved field. If present, it must be zero.

BITMAPINFOHEADER2 Field - usRecording

usRecording (USHORT)

Recording algorithm.

The format in which the bitmap data is recorded.

BRA_BOTTOMUP

Scan lines are recorded bottom to top. This is the default value.

BITMAPINFOHEADER2 Field - usRendering

usRendering (USHORT)

Halftoning algorithm.

The algorithm used to record bitmap data that has been digitally halftoned.

BRH_NOTHALFTONED

Bitmap data is not halftoned. This is the default value.

BRH_ERRORDIFFUSION

Error Diffusion or Damped Error Diffusion algorithm.

BRH_PANDA

Processing Algorithm for Non-coded Document Acquisition.

BRH_SUPERCIRCLE

Super Circle algorithm.

BITMAPINFOHEADER2 Field - cSize1

cSize1 (ULONG)

Size value 1.

If BRH_ERRORDIFFUSION is specified in *usRendering*, *cSize1* is the error damping as a percentage in the range 0 through 100. A value of 100% indicates no damping, and a value of 0% indicates that any errors are not diffused.

If BRH_PANDA or BRH_SUPERCIRCLE is specified, *cSize1* is the x dimension of the pattern used, in pels.

BITMAPINFOHEADER2 Field - cSize2

cSize2 (ULONG)

Size value 2.

If BRH_ERRORDIFFUSION is specified in *usRendering*, this parameter is ignored.

If BRH_PANDA or BRH_SUPERCIRCLE is specified, *cSize2* is the y dimension of the pattern used, in pels.

BITMAPINFOHEADER2 Field - ulColorEncoding

ulColorEncoding (ULONG)
Color encoding.

BCE_RGB

Each element in the color array is an RGB2 datatype. This is the default value.

BITMAPINFOHEADER2 Field - ulIdentifier

ulIdentifier (ULONG)
Reserved for application use.

BIT8

Defines eight independent BOOL values.

```
typedef use BIT8;
```

BIT16

Defines 16 independent BOOL values.

```
typedef use BIT16;
```

BIT32

Defines 32 independent BOOL values.

```
typedef use BIT32;
```

BOOL

Boolean.

Valid values are:

- FALSE, which is 0
- TRUE, which is 1

```
typedef unsigned long BOOL;
```

BOOKPAGEINFO

Notebook page information structure.

```
typedef struct _BOOKPAGEINFO {
    ULONG      cb;                /* Size of the page information structure. */
    ULONG      fl;                /* Flag indicating which page attributes are to be set. */
    BOOL       bLoadDlg;          /* Load dialog flag. */
    ULONG      ulPageData;        /* Data to associate with the notebook page. */
    HWND       hwndPage;          /* Handle to associate with the notebook page. */
    PFN        pfnPageDlgProc;    /* Dialog procedure. */
    ULONG      idPageDlg;         /* Dialog id. */
    HMODULE     hmodPageDlg;       /* Resource handle. */
    PVOID       pPageDlgCreateParam; /* Dialog create parameters. */
    PDLGTEMPLATE pdlgtPage;        /* Dialog template. */
    ULONG      cbStatusLine;       /* Length of status line text. */
    PSZ         pszStatusLine;     /* Status line text string. */
    HBITMAP     hbmMajorTab;        /* Major tab bit map handle. */
    HBITMAP     hbmMinorTab;        /* Minor tab bit map handle. */
    ULONG      cbMajorTab;         /* Length of major tab text. */
    PSZ         pszMajorTab;        /* Major tab text string. */
    ULONG      cbMinorTab;         /* Length of minor tab text. */
    PSZ         pszMinorTab;        /* Minor tab text string. */
    PVOID       pBidiInfo;         /* Reserved for bi-directional support. */
} BOOKPAGEINFO;

typedef BOOKPAGEINFO *PBOOKPAGEINFO;
```

BOOKPAGEINFO Field - cb

cb (ULONG)

Size of the page information structure.

BOOKPAGEINFO Field - fl

fl ([ULONG](#))

Flag indicating which page attributes are to be set.

- BFA_BIDIINFO Reserved for bi-directional support.
- BFA_MAJORTABBITMAP Set/query major tab bit map.
- BFA_MAJORTABTEXT Set/query major tab text.
- BFA_MINORTABBITMAP Set/query minor tab bit map.
- BFA_MINORTABTEXT Set/query minor tab text.
- BFA_PAGEDATA Set/query page data.
- BFA_PAGEFROMDLGRES Set/query page window handle from a dialog resource.
- BFA_PAGEFROMDLGTEMPLATE Set/query page window handle from a dialog template.
- BFA_PAGEFROMHWND Set/query page window handle.
- BFA_STATUSLINE Set/query status text.

BOOKPAGEINFO Field - bLoadDlg

bLoadDlg ([BOOL](#))

Load dialog flag.

- TRUE Load dialog immediately.
- FALSE Load dialog on page turn.

BOOKPAGEINFO Field - ulPageData

ulPageData ([ULONG](#))

Data to associate with the notebook page.

BOOKPAGEINFO Field - hwndPage

hwndPage ([HWND](#))

Handle to associate with the notebook page.

BOOKPAGEINFO Field - pfnPageDlgProc

pfnPageDlgProc (PFN)
Dialog procedure.

BOOKPAGEINFO Field - idPageDlg

idPageDlg (ULONG)
Dialog id.

BOOKPAGEINFO Field - hmodPageDlg

hmodPageDlg (HMODULE)
Resource handle.

BOOKPAGEINFO Field - pPageDlgCreateParam

pPageDlgCreateParam (PVOID)
Dialog create parameters.

BOOKPAGEINFO Field - pdlggtPage

pdlggtPage (PDLGTEMPLATE)
Dialog template.

BOOKPAGEINFO Field - cbStatusLine

cbStatusLine (ULONG)
Length of status line text.

BOOKPAGEINFO Field - pszStatusLine

pszStatusLine (PSZ)
Status line text string.

BOOKPAGEINFO Field - hbmMajorTab

hbmMajorTab (HBITMAP)
Major tab bit map handle.

BOOKPAGEINFO Field - hbmMinorTab

hbmMinorTab (HBITMAP)
Minor tab bit map handle.

BOOKPAGEINFO Field - cbMajorTab

cbMajorTab (ULONG)
Length of major tab text.

BOOKPAGEINFO Field - pszMajorTab

pszMajorTab (PSZ)
Major tab text string.

BOOKPAGEINFO Field - cbMinorTab

cbMinorTab (ULONG)
Length of minor tab text.

BOOKPAGEINFO Field - pszMinorTab

pszMinorTab ([PSZ](#))
Minor tab text string.

BOOKPAGEINFO Field - pBidilInfo

pBidilInfo ([PVOID](#))
Reserved for bi-directional support.

BOOKTEXT

Notebook data structure that contains text strings for notebook status lines and tabs. This data structure is used with the [BKM_QUERYSTATUSLINETEXT](#) and the [BKM_QUERYTABTEXT](#) messages only.

```
typedef struct _BOOKTEXT {
    PSZ      pString; /* Pointer to a string buffer. */
    ULONG    textLen; /* String length. */
} BOOKTEXT;

typedef BOOKTEXT *PBOOKTEXT;
```

BOOKTEXT Field - pString

pString ([PSZ](#))
Pointer to a string buffer.

Buffer in which the text string is to be placed. For the [BKM_QUERYSTATUSLINETEXT](#) message, this is the buffer in which the status line text is placed.

For the [BKM_QUERYTABTEXT](#) message, this is the buffer in which the tab text is placed.

BOOKTEXT Field - textLen

textLen ([ULONG](#))
String length.

Length of the text string. For the [BKM_QUERYSTATUSLINETEXT](#) message, this is the length of the status line text string.

For the [BKM_QUERYTABTEXT](#) message, this is the length of the tab text string.

BTNCDATA

Button-control-data structure.

```
typedef struct _BTNCDATA {  
    USHORT      cb;           /* Length of the control data in bytes. */  
    USHORT      fsCheckState; /* Check state of button. */  
    USHORT      fsHiliteState; /* Highlighting state of button. */  
    LHANDLE     hImage;       /* Resource handle for icon or bit map. */  
} BTNCDATA;  
  
typedef BTNCDATA *PBTNCDATA;
```

BTNCDATA Field - cb

cb ([USHORT](#))

Length of the control data in bytes.

This is the length of the control data for a button control.

BTNCDATA Field - fsCheckState

fsCheckState ([USHORT](#))

Check state of button.

This is the same value as returned by the [BM_QUERYCHECK](#) message and passed to the [BM_SETCHECK](#) message.

BTNCDATA Field - fsHiliteState

fsHiliteState ([USHORT](#))

Highlighting state of button.

This is the same value as returned by the [BM_QUERYHILITE](#) message and passed to the [BM_SETHILITE](#) message.

BTNCDATA Field - hImage

hImage ([LHANDLE](#))

Resource handle for icon or bit map.

BYTE

A byte.

```
typedef unsigned char BYTE;
```

CATCHBUF

Saved execution environment buffer.

```
typedef struct _CATCHBUF {  
    ULONG     reserved[4]; /* Save area. */  
} CATCHBUF;  
  
typedef CATCHBUF *PCATCHBUF;
```

CATCHBUF Field - reserved[4]

reserved[4] (ULONG)
Save area.

CDATE

Structure that contains date information for a data element in the details view of a container control.

```
typedef struct _CDATE {  
    UCHAR     day; /* Current day. */  
    UCHAR     month; /* Current month. */  
    USHORT    year; /* Current year. */  
} CDATE;  
  
typedef CDATE *PCDATE;
```

CDATE Field - day

day (UCHAR)
Current day.

CDATE Field - month

month (UCHAR)
Current month.

CDATE Field - year

year (USHORT)
Current year.

CHAR

Single-byte character.

#define CHAR char

CHARBUNDLE

Character-attributes bundle structure.

```
typedef struct _CHARBUNDLE {  
    LONG        lColor;           /* Character foreground color. */  
    LONG        lBackColor;       /* Character background color. */  
    USHORT      usMixMode;        /* Character foreground-mix mode. */  
    USHORT      usBackMixMode;    /* Character background-mix mode. */  
    USHORT      usSet;            /* Character set. */  
    USHORT      usPrecision;      /* Character precision. */  
    SIZEF       sizfxCell;        /* Character cell size. */  
    POINTL      ptlAngle;         /* Character angle. */  
    POINTL      ptlShear;         /* Character shear. */  
    USHORT      usDirection;      /* Character direction. */  
    USHORT      usTextAlign;      /* Text alignment. */  
    FIXED       fxExtra;          /* Character extra. */  
    FIXED       fxBreakExtra;     /* Character break extra. */  
} CHARBUNDLE;  
  
typedef CHARBUNDLE *PCHARBUNDLE;
```

CHARBUNDLE Field - lColor

IColor (LONG)
Character foreground color.

CHARBUNDLE Field - IBackColor

IBackColor (LONG)
Character background color.

CHARBUNDLE Field - usMixMode

usMixMode (USHORT)
Character foreground-mix mode.

CHARBUNDLE Field - usBackMixMode

usBackMixMode (USHORT)
Character background-mix mode.

CHARBUNDLE Field - usSet

usSet (USHORT)
Character set.

CHARBUNDLE Field - usPrecision

usPrecision (USHORT)
Character precision.

CHARBUNDLE Field - sizfxCell

sizeCell (SIZEF)
Character cell size.

CHARBUNDLE Field - ptlAngle

ptlAngle (POINTL)
Character angle.

CHARBUNDLE Field - ptlShear

ptlShear (POINTL)
Character shear.

CHARBUNDLE Field - usDirection

usDirection (USHORT)
Character direction.

CHARBUNDLE Field - usTextAlign

usTextAlign (USHORT)
Text alignment.

CHARBUNDLE Field - fxExtra

fxExtra (FIXED)
Character extra.

CHARBUNDLE Field - fxBreakExtra

fxBreakExtra ([FIXED](#))
Character break extra.

CLASSINFO

Class-information structure.

```
typedef struct _CLASSINFO {  
    ULONG     flClassStyle; /* Class-style flags. */  
    PFNWP     pfnWindowProc; /* Window procedure. */  
    ULONG     cbWindowData; /* Number of additional window words. */  
} CLASSINFO;  
  
typedef CLASSINFO *PCLASSINFO;
```

CLASSINFO Field - flClassStyle

flClassStyle ([ULONG](#))
Class-style flags.

CLASSINFO Field - pfnWindowProc

pfnWindowProc ([PFNWP](#))
Window procedure.

CLASSINFO Field - cbWindowData

cbWindowData ([ULONG](#))
Number of additional window words.

CNRDRAGINFO

Structure that contains information about a direct manipulation event that is occurring over the container. The information specified for this structure depends on the container notification code with which it is used. The differences are specified in the following field descriptions. The applicable notification codes are:

- [CN_DRAGAFTER](#)
- [CN_DRAGLEAVE](#)

- [CN_DRAGOVER](#)
- [CN_DROP](#)
- [CN_DROPHELP](#)

```
typedef struct _CNRDRAGINFO {
    PDRAGINFO      pDragInfo; /* Pointer to a DRAGINFO structure. */
    PRECORDCORE    pRecord; /* Pointer to a RECORDCORE structure. */
} CNRDRAGINFO;

typedef CNRDRAGINFO *PCNRDRAGINFO;
```

CNRDRAGINFO Field - pDragInfo

pDragInfo ([PDRAGINFO](#))
Pointer to a [DRAGINFO](#) structure.

CNRDRAGINFO Field - pRecord

pRecord ([PRECORDCORE](#))
Pointer to a [RECORDCORE](#) structure.

The structure that is pointed to depends on the notification code being used.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of [PRECORDCORE](#) in all applicable data structures and messages. For the [CN_DRAGAFTER](#) notification code, this field contains a pointer to the [RECORDCORE](#) structure after which ordered target emphasis is drawn. If ordered target emphasis is applied above the first record in item order, the CMA_FIRST attribute is returned.

For the [CN_DRAGLEAVE](#) notification code, this field is NULL.

For the [CN_DRAGOVER](#), [CN_DROP](#), and [CN_DROPHELP](#) notification codes, this field contains a pointer to a container record over which direct manipulation occurred. This field has a value of NULL if the direct manipulation event occurs over white space.

CNRDRAGINIT

Structure that contains information about a direct manipulation event that is initiated in a container. This structure is used with the [CN_INITDRAG](#) notification code only.

```
typedef struct _CNRDRAGINIT {
    HWND          hwndCnr; /* Container control handle. */
    PRECORDCORE    pRecord; /* Pointer to the RECORDCORE where direct manipulation started. */
    LONG          x; /* X-coordinate of the pointer of the pointing device in desktop coordinates. */
    LONG          y; /* Y-coordinate of the pointer of the pointing device in desktop coordinates. */
    LONG          cx; /* X-offset from the hot spot of the pointer of the pointing device (in pels) to t
    LONG          cy; /* Y-offset from the hot spot of the pointer of the pointing device (in pels) to t
} CNRDRAGINIT;

typedef CNRDRAGINIT *PCNRDRAGINIT;
```

CNRDRAGINIT Field - hwndCnr

hwndCnr (HWND)
Container control handle.

CNRDRAGINIT Field - pRecord

pRecord (RECORDCORE)
Pointer to the RECORDCORE where direct manipulation started.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of RECORDCORE in all applicable data structures and messages.

The *pRecord* field can have one of the following values:

- NULL
Direct manipulation started over white space.
 - Other
Container record over which direct manipulation started.
-

CNRDRAGINIT Field - x

x (LONG)
X-coordinate of the pointer of the pointing device in desktop coordinates.

CNRDRAGINIT Field - y

y (LONG)
Y-coordinate of the pointer of the pointing device in desktop coordinates.

CNRDRAGINIT Field - cx

cx (LONG)
X-offset from the hot spot of the pointer of the pointing device (in pels) to the record origin.

CNRDRAGINIT Field - cy

cy ([LONG](#))

Y-offset from the hot spot of the pointer of the pointing device (in pels) to the record origin.

CNRDRAWITEMINFO

Structure that contains information about the container item being drawn. This structure is used with the [WM_DRAWITEM](#) (in [Container Controls](#)) message only.

```
typedef struct _CNRDRAWITEMINFO {
    PRECORDCORE    pRecord;    /* Pointer to the RECORDCORE structure for the record being drawn. */
    PFIELDINFO     pFieldInfo; /* Pointer to the FIELDINFO structure for the container column being drawn in t
} CNRDRAWITEMINFO;

typedef CNRDRAWITEMINFO *PCNRDRAWITEMINFO;
```

CNRDRAWITEMINFO Field - pRecord

pRecord ([PRECORDCORE](#))

Pointer to the [RECORDCORE](#) structure for the record being drawn.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

CNRDRAWITEMINFO Field - pFieldInfo

pFieldInfo ([PFIELDINFO](#))

Pointer to the [FIELDINFO](#) structure for the container column being drawn in the details view.

For all other views, this field is NULL.

CNREDITDATA

Structure that contains information about the direct editing of container text. The information specified for this structure depends on the container notification code or message with which it is used. The differences are specified in the following field descriptions. The applicable notification codes and message are:

- [CN_BEGINEDIT](#)
- [CN_ENDEDIT](#)
- [CN_REALLOCPSZ](#)
- [CM_OPENEDIT](#)

```
typedef struct _CNREDITDATA {
    ULONG      cb;           /* Structure size. */
    HWND       hwndCnr;      /* Container window handle. */
    PRECORDCORE pRecord;     /* Pointer to a RECORDCORE data structure, or NULL. */
    PFIELDINFO  pFieldInfo;  /* Pointer to a FIELDINFO data structure, or NULL. */
    PSZ        *ppszText;    /* Pointer to a PSZ text string. */
    ULONG      cbText;       /* Number of bytes in the text string. */
    ULONG      id;           /* ID of the window to be edited. */
} CNREDITDATA;

typedef CNREDITDATA *PCNREDITDATA;
```

CNREDITDATA Field - cb

cb ([ULONG](#))
Structure size.

The size (in bytes) of the CNREDITDATA data structure.

CNREDITDATA Field - hwndCnr

hwndCnr ([HWND](#))
Container window handle.

CNREDITDATA Field - pRecord

pRecord ([PRECORDCORE](#))
Pointer to a [RECORDCORE](#) data structure, or NULL.

This field is NULL if container titles are to be edited.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages. For the [CN_BEGINEDIT](#), [CN_ENDEDIT](#), and [CN_REALLOCPSZ](#) notification codes, this field is a pointer to the edited [RECORDCORE](#) data structure.

For the [CM_OPENEDIT](#) message, this field is a pointer to the [RECORDCORE](#) data structure to be edited.

CNREDITDATA Field - pFieldInfo

pFieldInfo (PFIELDINFO)

Pointer to a **FIELDINFO** data structure, or NULL.

Pointer to a **FIELDINFO** data structure if the current view is the details view and the user is not editing the container title. Otherwise, this field is NULL.

If the current view is the details view:

- For the **CN_BEGINEDIT**, **CN_ENDEDIT**, and **CN_REALLOCPSZ** notification codes, this field contains a pointer to the **FIELDINFO** structure being edited.
- For the **CM_OPENEDIT** message, this field is a pointer to the **FIELDINFO** data structure to be edited.

CNREDITDATA Field - ppszText

ppszText (PSZ *)

Pointer to a **PSZ** text string.

For the **CN_BEGINEDIT** and **CN_REALLOCPSZ** notification codes, this field is a pointer to the current **PSZ** text string.

For the **CN_ENDEDIT** notification code, this field is a pointer to the new **PSZ** text string.

For the **CM_OPENEDIT** message, this field is NULL.

CNREDITDATA Field - cbText

cbText (ULONG)

Number of bytes in the text string.

For the **CN_BEGINEDIT** notification code, this field is 0.

For the **CN_ENDEDIT** and **CN_REALLOCPSZ** notification codes, this field is the number of bytes in the new text string.

For the **CM_OPENEDIT** message, this field is 0.

CNREDITDATA Field - id

id (ULONG)

ID of the window to be edited.

The ID can be one of the following:

CID_CNRTITLEWND

Title window.

CID_LEFTDVWND

Left details view window; default if unsplit window.

CID_RIGHTDVWND

Right details view window.

CID_LEFTCOLTITLEWND
Left details view column headings window; default if unsplit window.

CID_RIGHTCOLTITLEWND
Right details view column headings window.

An application-defined container-ID
Container window.

CNRINFO

Structure that contains information about the container.

```
typedef struct _CNRINFO {
    ULONG cb; /* Structure size. */
    PVOID pSortRecord; /* Pointer to the comparison function for sorting container records, or NULL. */
    PFIELDINFO pFieldInfoLast; /* Pointer to the last column in the left window of the split details view. */
    PFIELDINFO pFieldInfoObject; /* Pointer to a column that represents an object in the details view. */
    PSZ pszCnrTitle; /* Title text, or NULL. */
    ULONG flWindowAttr; /* Window attributes. */
    POINTL ptlOrigin; /* Workspace origin. */
    ULONG cDelta; /* Threshold. */
    ULONG cRecords; /* Number of records. */
    SIZEL slBitmapOrIcon; /* Icon/bit-map size. */
    SIZEL slTreeBitmapOrIcon; /* Icon/bit-map size. */
    HBITMAP hbmExpanded; /* Bit-map handle. */
    HBITMAP hbmCollapsed; /* Bit-map handle. */
    HPOINTER hpPtrExpanded; /* Icon handle. */
    HPOINTER hpPtrCollapsed; /* Icon handle. */
    LONG cyLineSpacing; /* Vertical space. */
    LONG cxTreeIndent; /* Horizontal space. */
    LONG cxTreeLine; /* Line width. */
    ULONG cFields; /* Number of columns. */
    LONG xVertSplitbar; /* Split bar position. */
} CNRINFO;

typedef CNRINFO *PCNRINFO;
```

CNRINFO Field - cb

cb (**ULONG**)
Structure size.

The size (in bytes) of the CNRINFO data structure.

CNRINFO Field - pSortRecord

pSortRecord (**PVOID**)
Pointer to the comparison function for sorting container records, or NULL.

If NULL, which is the default condition, no sorting is performed. Sorting occurs only during record insertion and when changing the value of this field. The third parameter of the comparison function, *pStorage*, must be NULL. See "CM_SORTRECORD" in the *Presentation Manager Programming Reference* for a further description of the comparison function.

CNRINFO Field - pFieldInfoLast

pFieldInfoLast ([PFIELDINFO](#))
Pointer to the last column in the left window of the split details view, or NULL.

The default is NULL, which causes all columns to be positioned in the left window.

CNRINFO Field - pFieldInfoObject

pFieldInfoObject ([PFIELDINFO](#))
Pointer to a column that represents an object in the details view.

The data for this [FIELDINFO](#) structure must contain icons or bit maps. In-use emphasis is applied to this column of icons or bit maps only. The default is the leftmost column in the unsplit details view or the leftmost column in the left window of the split details view.

CNRINFO Field - pszCnrTitle

pszCnrTitle ([PSZ](#))
Title text, or NULL.

Text for the container title. The application can change the container title during processing. The default is NULL.

CNRINFO Field - flWindowAttr

flWindowAttr ([ULONG](#))
Window attributes.

Consists of container window attributes.

- Specify one of the following container views, which determine the presentation format of items in a container:

CV_ICON	In the icon view, the container items are represented as icon/text or bit-map/text pairs, with text beneath the icons or bit maps. This is the default view. This view can be combined with the CV_MINI style bit by using an OR operator ().
CV_NAME	In the name view, the container items are represented as icon/text or bit-map/text pairs, with text to the right of the icons or bit maps. This view can be combined with the CV_MINI and CV_FLOW style bits by using OR operators ().
CV_TEXT	In the text view, the container items are displayed as a list of text strings. This view can be combined with the CV_FLOW style bit by using an OR operator ().

CV_TREE

In the tree view, the container items are represented in a hierarchical manner. The tree view has three forms, which are defined in the following list. If you specify CV_TREE by itself, the tree icon view is used.

- Tree icon view

The tree icon view is specified by using a logical OR operator to combine the tree view with the icon view (CV_TREE | CV_ICON). Container items in this view are represented as icon/text pairs or bit-map/text pairs, with text to the right of the icons or bit maps. Also, a collapsed or expanded icon or bit map is displayed to the left of parent items. If this icon or bit map is a *collapsed* icon or bit map, selecting it will cause the parent item to be expanded so that its child items are displayed below it. If this icon or bit map is an *expanded* icon or bit map, selecting it will cause the parent's child items to be removed from the display. The default collapsed and expanded bit maps provided by the container use a plus sign (+) and a minus sign (-), respectively, to indicate that items can be added to or subtracted from the display.

- Tree name view

The tree name view is specified by using a logical OR operator to combine the tree view with the name view (CV_TREE | CV_NAME). Container items in this view are displayed as either icon/text pairs or bit-map/text pairs, with text to the right of the icons or bit maps. However, the indicator that represents whether an item can be collapsed or expanded, such as a plus or minus sign, is included in the icon or bit map that represents that item, not in a separate icon or bit map as in the tree icon and tree text views. The container control does not provide default collapsed and expanded bit maps for the tree name view.

- Tree text view

The tree text view is specified by using a logical OR operator to combine the tree view with the text view (CV_TREE | CV_TEXT). Container items in this view are displayed as a list of text strings. As in the tree icon view, a collapsed or expanded icon or bit map is displayed to the left of parent items.

CV_DETAIL

In the details view, the container items are presented in columns. Each column can contain icons or bit maps, text, numbers, dates, or times.

- Specify one or both of the following view styles by using an OR operator (|) to combine them with the specified view. These view styles are optional.

CV_MINI

Produces a mini-icon whose size is based on the Presentation Manager (PM) SV_CYMENU system value to produce a device-dependent mini-icon.

The CV_MINI view style bit is ignored when:

- The text view (CV_TEXT), tree view (CV_TREE), or details view (CV_DETAIL) are displayed
- The CCS_MINIRECORDCORE style bit is specified.

If this style bit is not specified and the icon view (CV_ICON) or name view (CV_NAME) is used, the default, regular-sized icon is used. The size of regular-sized icons is based on the value in the *s/BitmapOrIcon* field of the CNRINFO data structure. If this field is equal to 0, the PM SV_CXICON and SV_CYICON system values for width and height, respectively, are used. Icon sizes are consistent with PM-defined icon sizes for all devices.

CV_FLOW

Dynamically arranges container items in columns in the name and text views. These are called flowed name and flowed text views. If this style bit is set for the name view (CV_NAME) or text view (CV_TEXT), the container items are placed in a single column until the bottom of the client area is reached. The next container item is placed in the adjacent column to the right of the filled column. This process is repeated until all of the container items are positioned in the container. The width of each column is determined by the longest text string in that column. The size of the window determines the depth of the client area.

If this style bit is not specified, the default condition for the name and text views is to vertically fill the container in a single column without flowing the container items. If this style bit is set for the icon view (CV_ICON) or details view (CV_DETAIL), it is ignored.

- Specify either of the following to indicate whether the container will display icons or bit maps:

CA_DRAWICON

Icons are used for the icon, name, tree, or details views. This is the default. This container attribute should be used with the *hptrIcon* and *hptrMiniIcon* fields of the [RECORDCORE](#) data structure.

CA_DRAWBITMAP

Bit maps are used for the icon, name, tree, or details views. This container attribute can be used with the *hbmBitmap* and *hbmMiniBitmap* fields of the [RECORDCORE](#) data structure.

Notes

1. If both the CA_DRAWICON and CA_DRAWBITMAP attributes are specified, the CA_DRAWICON attribute is used.
 2. If the CCS_MINIRECORDCORE style bit is specified when a container is created, the *hptrIcon* field of the [MINIRECORDCORE](#) data structure is used.
- Specify one of the following attributes to provide target emphasis for the name, text, and details views. If neither ordered nor mixed target emphasis is specified, the emphasis is drawn around the record.

CA_ORDEREDTARGETEMPH

Shows where a container record can be dropped during direct manipulation by drawing a line beneath the record. Ordered target emphasis does not apply to the icon and tree views.

CA_MIXEDTARGETEMPH

Shows where a container record can be dropped during direct manipulation either by drawing a line between two items or by drawing lines around the container record. Mixed target emphasis does not apply to the icon and tree views.

- Specify the following attribute to draw lines that show the relationship between items in the tree view.

CA_TREELINE

Shows the relationship between all items in the tree view.

- Specify the following to draw container records, paint the background of the container, or both:

CA_OWNERDRAW

Ownerdraw for the container, which allows the application to draw container records.

CA_OWNERPAINTBACKGROUND

Allows the application to subclass the container and paint the background. If specified, and the container is subclassed, the application receives the CM_PAINTBACKGROUND message in the subclass procedure. Otherwise, the container paints the background using the color specified by SYSCLR_WINDOW, which can be changed by using the PP_BACKGROUND_COLOR or PP_BACKGROUND_COLOR_INDEX presentation parameter in the WM_PRESPARAMCHANGED (in Container Controls)

- Specify the following if the container is to have a title:

CA_CONTAINERTITLE

Allows you to include a container title. The default is no container title.

- Specify one or both of the following container title attributes. These are valid only if the CA_CONTAINERTITLE attribute is specified.

CA_TITLEREADONLY

Prevents the container title from being edited directly. The default is to allow the container title to be edited.

CA_TITLESEPARATOR

Puts a separator line between the container title and the records beneath it. The default is no separator line.

- Specify one of the following to position the container title. These are valid only if the CA_CONTAINERTITLE attribute is specified.

CA_TITLECENTER

Centers the container title. This is the default.

CA_TITLELEFT

Left-justifies the container title.

CA_TITLERIGHT

Right-justifies the container title.

- Specify the following to display column headings in the details view:

CA_DETAILSVIEWTITLES

Allows you to include column headings in the details view. The default is no column headings.

CNRINFO Field - ptlOrigin

ptlOrigin ([POINTL](#))

Workspace origin.

Lower-left origin of the workspace in virtual coordinates, used in the icon view. The default origin is (0,0).

CNRINFO Field - cDelta

cDelta ([ULONG](#))

Threshold.

An application-defined threshold, or number of records, from either end of the list of available records. Used when a container needs to handle large amounts of data. The default is 0. See the *OS/2 Programming Guide* for more information about specifying deltas.

CNRINFO Field - cRecords

cRecords ([ULONG](#))

Number of records.

The number of records in the container. Initially, this field is 0.

CNRINFO Field - slBitmapOrIcon

slBitmapOrIcon ([SIZEL](#))

Icon/bit-map size.

The size (in pels) of icons or bit maps. The default is the system size.

CNRINFO Field - slTreeBitmapOrIcon

slTreeBitmapOrIcon ([SIZEL](#))
Icon/bit-map size.

The size (in pels) of the expanded and collapsed icons or bit maps used in the tree icon and tree text views.

CNRINFO Field - hbmExpanded

hbmExpanded ([HBITMAP](#))
Bit-map handle.

The handle of the bit map to be used to represent an expanded parent item in the tree icon and tree text views. If neither an icon handle (see *hptrExpanded*) nor a bit-map handle is specified, a default bit map with a minus sign (-) is provided.

CNRINFO Field - hbmCollapsed

hbmCollapsed ([HBITMAP](#))
Bit-map handle.

The handle of the bit map to be used to represent a collapsed parent item in the tree icon and tree text views. If neither an icon handle (see *hptrCollapsed*) nor a bit-map handle is specified, a default bit map with a plus sign (+) is provided.

CNRINFO Field - hptrExpanded

hptrExpanded ([HPOINTER](#))
Icon handle.

The handle of the icon to be used to represent an expanded parent item in the tree icon and tree text views. If neither an icon handle nor a bit-map handle (see *hbmExpanded*) is specified, a default bit map with a minus sign (-) is provided.

CNRINFO Field - hptrCollapsed

hptrCollapsed ([HPOINTER](#))
Icon handle.

The handle of the icon to be used to represent a collapsed parent item in the tree icon and tree text views. If neither an icon handle nor a bit-map handle (see *hbmCollapsed*) is specified, a default bit map with a plus sign (+) is provided.

CNRINFO Field - cyLineSpacing

cyLineSpacing (LONG)
Vertical space.

The amount of vertical space (in pels) between the records. If you specify a value that is less than 0, a default value is used.

CNRINFO Field - cxTreeIndent

cxTreeIndent (LONG)
Horizontal space.

The amount of horizontal space (in pels) between levels in the tree view. If you specify a value that is less than 0, a default value is used.

CNRINFO Field - cxTreeLine

cxTreeLine (LONG)
Line width.

The width of the lines (in pels) that show the relationship between tree items. If you specify a value that is less than 0, a default value is used. Also, if the CA_TREELINE container attribute of the *flWindowAttr* field is not specified, these lines are not drawn.

CNRINFO Field - cFields

cFields (ULONG)
Number of columns.

The number of **FIELDINFO** structures in the container. Initially, this field is 0.

CNRINFO Field - xVertSplitbar

xVertSplitbar (LONG)
Split bar position.

The initial position of the split bar relative to the container used in the details view. If this value is less than 0, the split bar is not used. The default value is negative one (-1).

CNRLAZYDRAGINFO

Container lazy drag information.

```
typedef struct _CNRLAZYDRAGINFO {
    PDRAGINFO      pDragInfo; /* Pointer to the DRAGINFO structure. */
    PRECORDCORE    pRecord;   /* Pointer to a container RECORDCORE structure. */
    HWND           hwndTarget; /* Handle of the target winddow that the lazy drag set was dropped on. */
} CNRLAZYDRAGINFO;

typedef CNRLAZYDRAGINFO *PCNRLAZYDRAGINFO;
```

CNRLAZYDRAGINFO Field - pDragInfo

pDragInfo ([PDRAGINFO](#))
Pointer to the [DRAGINFO](#) structure.

CNRLAZYDRAGINFO Field - pRecord

pRecord ([PRECORDCORE](#))
Pointer to a container [RECORDCORE](#) structure.

A value of NULL indicates that the lazy drag set was dropped over whitespace in the container. Any other value indicates that the lazy drag set was dropped on the record specified by this field.

CNRLAZYDRAGINFO Field - hwndTarget

hwndTarget ([HWND](#))
Handle of the target winddow that the lazy drag set was dropped on.

COLOR

Color value.

```
typedef LONG COLOR;
```

CONVCONTEXT

Dynamic-data-exchange conversation context structure.

```
typedef struct _CONVCONTEXT {
    ULONG      cb;           /* Length of structure. */
    ULONG      fsContext;    /* Options. */
    ULONG      idCountry;    /* Country code. */
    ULONG      usCodepage;   /* Code-page identity. */
    ULONG      usLangID;     /* Language. */
    ULONG      usSubLangID;  /* Sub-language. */
} CONVCONTEXT;

typedef CONVCONTEXT *PCONVCONTEXT;
```

CONVCONTEXT Field - cb

cb (ULONG)
Length of structure.

This must be set to the length of the CONVCONTEXT structure.

CONVCONTEXT Field - fsContext

fsContext (ULONG)
Options.

DDECTXT_CASESENSITIVE
All strings in this conversation are case sensitive.

CONVCONTEXT Field - idCountry

idCountry (ULONG)
Country code.

CONVCONTEXT Field - usCodepage

usCodepage (ULONG)
Code-page identity.

CONVCONTEXT Field - usLangID

usLangID (**ULONG**)
Language.

Zero is valid and means no language information.

CONVCONTEXT Field - usSubLangID

usSubLangID (**ULONG**)
Sub-language.

Zero is valid and means no sub-language information.

CREATESTRUCT

Create-window data structure.

```
typedef struct _CREATESTRUCT {
    PVOID    pPresParams; /* Presentation parameters. */
    PVOID    pCtlData;    /* Control data. */
    ULONG    id;          /* Window identifier. */
    HWND     hwndInsertBehind; /* Window behind which the window is to be placed. */
    HWND     hwndOwner;   /* Window owner. */
    LONG     cy;          /* Window height. */
    LONG     cx;          /* Window width. */
    LONG     y;           /* Y-coordinate of origin. */
    LONG     x;           /* X-coordinate of origin. */
    ULONG    flStyle;     /* Window style. */
    PSZ      pszText;     /* Window text. */
    PSZ      pszClassName; /* Registered window class name. */
    HWND     hwndParent;  /* Parent window handle. */
} CREATESTRUCT;

typedef CREATESTRUCT *PCREATESTRUCT;
```

CREATESTRUCT Field - pPresParams

pPresParams (**PVOID**)
Presentation parameters.

CREATESTRUCT Field - pCtlData

pCtlData (**PVOID**)

Control data.

CREATESTRUCT Field - id

id ([ULONG](#))
Window identifier.

CREATESTRUCT Field - hwndInsertBehind

hwndInsertBehind ([HWND](#))
Window behind which the window is to be placed.

CREATESTRUCT Field - hwndOwner

hwndOwner ([HWND](#))
Window owner.

CREATESTRUCT Field - cy

cy ([LONG](#))
Window height.

CREATESTRUCT Field - cx

cx ([LONG](#))
Window width.

CREATESTRUCT Field - y

y ([LONG](#))
Y-coordinate of origin.

CREATESTRUCT Field - x

x ([LONG](#))
X-coordinate of origin.

CREATESTRUCT Field - flStyle

flStyle ([ULONG](#))
Window style.

CREATESTRUCT Field - pszText

pszText ([PSZ](#))
Window text.

CREATESTRUCT Field - pszClassName

pszClassName ([PSZ](#))
Registered window class name.

CREATESTRUCT Field - hwndParent

hwndParent ([HWND](#))
Parent window handle.

CSBITMAPDATA

This is the bitmap data structure for the circular slider buttons.

```
typedef struct _CSBITMAPDATA {
    HBITMAP    hbmLeftUp;
    HBITMAP    hbmLeftDown;
    HBITMAP    hbmRightUp;
    HBITMAP    hbmRightDown;
} CSBITMAPDATA;

typedef CSBITMAPDATA *PCSBITMAPDATA;
```

CSBITMAPDATA Field - hbmLeftUp

hbmLeftUp ([HBITMAP](#))
Handle to the "up" position bit map for the button on the left.

CSBITMAPDATA Field - hbmLeftDown

hbmLeftDown ([HBITMAP](#))
Handle to "down" position bit map for the button on the left.

CSBITMAPDATA Field - hbmRightUp

hbmRightUp ([HBITMAP](#))
Handle to the "up" position bit map for the button on the right.

CSBITMAPDATA Field - hbmRightDown

hbmRightDown ([HBITMAP](#))
Handle to the "down" position bit map for the button on the right.

CTIME

Structure that contains time information for a data element in the details view of a container control.

```
typedef struct _CTIME {
    UCHAR    hours;        /* Current hour. */
    UCHAR    minutes;     /* Current minute. */
};
```



```
    UCHAR    seconds;    /* Current second. */
    UCHAR    ucReserved; /* Reserved. */
} CTIME;

typedef CTIME *PCTIME;
```

CTIME Field - hours

hours (UCHAR)
Current hour.

CTIME Field - minutes

minutes (UCHAR)
Current minute.

CTIME Field - seconds

seconds (UCHAR)
Current second.

CTIME Field - ucReserved

ucReserved (UCHAR)
Reserved.

CTLCOLOR

Structure that contains a control color index and value pair.

```
typedef struct _CTLCOLOR {
    LONG    clrIndex; /* Control color index. */
    ULONG    clrValue; /* Control color value. */
} CTLCOLOR;

typedef CTLCOLOR *PCTLCOLOR;
```

CTLCOLOR Field - clrIndex

clrIndex (LONG)

Control color index.

This value can be any of the following control color index constants.

CCI_FOREGROUND
Foreground color index.
CCI_FOREGROUNDREADONLY
Output text color index.
CCI_BACKGROUND
Background color index.
CCI_BACKGROUNDDDIALOG
Dialog background color index.
CCI_DISABLEDFOREGROUND
Disabled foreground color index.
CCI_DISABLEDFOREGROUNDREADONLY
Disabled output text color index.
CCI_DISABLEDBACKGROUND
Disabled background color index.
CCI_DISABLEDBACKGROUNDDDIALOG
Disabled dialog background color index.
CCI_HIGHLIGHTFOREGROUND
Highlighted foreground color index.
CCI_HIGHLIGHTBACKGROUND
Highlighted background color index.
CCI_HIGHLIGHTBACKGROUNDDDIALOG
Highlighted dialog background color index.
CCI_INACTIVEFOREGROUND
Inactive foreground color index.
CCI_INACTIVEFOREGROUNDDDIALOG
Inactive dialog foreground color index.
CCI_INACTIVEBACKGROUND
Inactive background color index.
CCI_INACTIVEBACKGROUNDTEXT
Inactive background text color index.
CCI_ACTIVEFOREGROUND
Active foreground color index.
CCI_ACTIVEFOREGROUNDDDIALOG
Active dialog foreground color index.
CCI_ACTIVEBACKGROUND
Active background color index.
CCI_ACTIVEBACKGROUNDTEXT
Active background text color index.
CCI_PAGEBACKGROUND
Page background color index.
CCI_PAGEFOREGROUND
Page foreground color index.
CCI_FIELDBACKGROUND
Field background color index.
CCI_BORDER
Border color index.
CCI_BORDERLIGHT
Border light color index.
CCI_BORDERDARK
Border dark color index.
CCI_BORDER2
Second border color index.
CCI_BORDER2LIGHT
Second border light color index.
CCI_BORDER2DARK
Second border dark color index.
CCI_BORDERDEFAULT

	Border default color index.
CCI_BUTTONBACKGROUND	Button background color index.
CCI_BUTTONFOREGROUND	Button foreground color index.
CCI_BUTTONBORDERLIGHT	Button border light color index.
CCI_BUTTONBORDERDARK	Button border dark color index.
CCI_ARROW	Arrow color index.
CCI_DISABLEDARROW	Disabled arrow color index.
CCI_ARROWBORDERLIGHT	Arrow border light color index.
CCI_ARROWBORDERDARK	Arrow border dark color index.
CCI_CHECKLIGHT	Check light color index.
CCI_CHECKMIDDLE	Check middle color index.
CCI_CHECKDARK	Check dark color index.
CCI_ICONFOREGROUND	Icon foreground color index.
CCI_ICONBACKGROUND	Icon background color index.
CCI_ICONBACKGROUNDDESKTOP	Icon background desktop color index.
CCI_ICONHILITEFOREGROUND	Icon highlighted foreground color index.
CCI_ICONHILITEBACKGROUND	Icon highlighted background color index.
CCI_MAJORTABFOREGROUND	Major tab foreground color index.
CCI_MAJORTABBACKGROUND	Major tab background color index.
CCI_MINORTABFOREGROUND	Minor tab foreground color index.
CCI_MINORTABBACKGROUND	Minor tab background color index.

CTLCOLOR Field - clrValue

clrValue (ULONG)

Control color value.

This field can contain any valid RGB value or any of the following SYSCLR_* constants.

When CTLCOLORS is used with WinQueryControlColors, CCV_NOTFOUND can be returned to this field, which indicates no control color value was found for the specified color index.

When CTLCOLORS is used with WinSetControlColors, you can use CCV_IGNORE in this field to indicate you do not want to set a color value for this index. You can reset the color for an index to the default by specifying CCV_DEFAULT.

SYSCLR_SHADOWHILITEBGND	System color for shadow highlighted background.
SYSCLR_SHADOWHILITEFGND	System color for shadow highlighted foreground.
SYSCLR_SHADOWTEXT	System color for shadow text.
SYSCLR_ENTRYFIELD	System color for entry field.
SYSCLR_MENUDISABLEDTEXT	System color for disabled menu text.

SYSCLR_MENUHILITE
System color for highlighted menu text.

SYSCLR_MENUHILITEBGND
System color for highlighted menu background.

SYSCLR_PAGEBACKGROUND
System color for page background.

SYSCLR_FIELDBACKGROUND
System color for field background.

SYSCLR_BUTTONLIGHT
System color for light button.

SYSCLR_BUTTONMIDDLE
System color for middle button.

SYSCLR_BUTTONDARK
System color for dark button.

SYSCLR_BUTTONDEFAULT
System color for default button.

SYSCLR_TITLEBOTTOM
System color for title bottom.

SYSCLR_SHADOW
System color for shadow.

SYSCLR_ICONTEXT
System color for icon text.

SYSCLR_DIALOGBACKGROUND
System color for dialog background.

SYSCLR_HILITEFOREGROUND
System color for highlighted foreground.

SYSCLR_HILITEBACKGROUND
System color for highlighted background.

SYSCLR_TITLETEXT
System color for title text.

SYSCLR_INACTIVETITLETEXTBGND
System color for inactive title text background.

SYSCLR_ACTIVETITLETEXTBGND
System color for active title text background.

SYSCLR_INACTIVETITLETEXT
System color for inactive title text.

SYSCLR_ACTIVETITLETEXT
System color for active title text.

SYSCLR_OUTPUTTEXT
System color for output text.

SYSCLR_WINDOWSTATICTEXT
System color for static window text.

SYSCLR_SCROLLBAR
System color for scroll bar.

SYSCLR_BACKGROUND
System color for background.

SYSCLR_ACTIVETITLE
System color for active title.

SYSCLR_INACTIVETITLE
System color for inactive title.

SYSCLR_MENU
System color for a menu.

SYSCLR_MENUTEXT
System color for menu text.

SYSCLR_WINDOW
System color for a window.

SYSCLR_WINDOWTEXT
System color for window text.

SYSCLR_WINDOWFRAME
System color for a window frame.

SYSCLR_ACTIVEBORDER
System color for active border.

SYSCLR_INACTIVEBORDER
System color for inactive border.

SYSCLR_APPWORKSPACE
System color for application work space.

SYSCLR_HELPBACKGROUND
System color for help background.

SYSCLR_HELPTEXT
System color for help text.

SYSCLR_HELPHILITE
System color for help highlighting.

Control-Data

Pointer to class-specific control data, beginning with a value conforming to a [USHORT](#) data type, which specifies the overall length of the data.

There are several different types of control-data structures:

BTNCDATA	Button control data
ENTRYFDATA	Entry field control data
FRAMECDATA	Frame control data
MLECTLDATA	Multi-line entry field control data
SBCDATA	Scroll bar control data.

CURSORINFO

Cursor-information structure.

```
typedef struct _CURSORINFO {
    HWND    hwnd;      /* Window handle. */
    LONG    x;         /* X-coordinate. */
    LONG    y;         /* Y-coordinate. */
    LONG    cx;        /* Cursor width. */
    LONG    cy;        /* Cursor height. */
    ULONG    fs;       /* Options. */
    RECT    rclClip;   /* Cursor box. */
} CURSORINFO;

typedef CURSORINFO *PCURSORINFO;
```

CURSORINFO Field - hwnd

hwnd ([HWND](#))
Window handle.

CURSORINFO Field - x

x ([LONG](#))
X-coordinate.

CURSORINFO Field - y

y (**LONG**)
Y-coordinate.

CURSORINFO Field - cx

cx (**LONG**)
Cursor width.

CURSORINFO Field - cy

cy (**LONG**)
Cursor height.

CURSORINFO Field - fs

fs (**ULONG**)
Options.

CURSORINFO Field - rclClip

rclClip (**RECTL**)
Cursor box.

DDEINIT

Dynamic-data-exchange initiation structure.

```
typedef struct _DDEINIT {
    ULONG      cb;           /* Length of structure. */
    PSZ        pszAppName;   /* Application name. */
    PSZ        pszTopic;     /* Topic. */
    ULONG      offConvContext; /* Conversation context. */
} DDEINIT;

typedef DDEINIT *PDDEINIT;
```

DDEINIT Field - cb

cb ([ULONG](#))

Length of structure.

This must be set to the length of the DDEINIT structure.

DDEINIT Field - pszAppName

pszAppName ([PSZ](#))

Application name.

Pointer to name of the server application.

Application names must not contain slashes or backslashes. These characters are reserved for future use in network implementations.

DDEINIT Field - pszTopic

pszTopic ([PSZ](#))

Topic.

Pointer to name of the topic.

DDEINIT Field - offConvContext

offConvContext ([ULONG](#))

Conversation context.

Offset to a [CONVCONTEXT](#) structure.

DDESTRUCT

Dynamic-data-exchange control structure.

```
typedef struct _DDESTRUCT {  
    ULONG         cbData;          /* Length of the data. */  
};
```

```

USHORT    fsStatus;        /* Status of the data exchange. */
USHORT    usFormat;        /* DDE data format. */
USHORT    offszItemName;    /* Offset to item name. */
USHORT    offabData;        /* Offset to beginning of data. */
} DDESTRUCT;

typedef DDESTRUCT *PDDESTRUCT;

```

DDE formats, specified in the *usFormat* parameter, are registered with the atom manager, using the system atom table. The predefined DDE formats are guaranteed not to conflict with the values returned by the atom manager.

Applications can define their own data formats; however, each nonstandard DDE format must have a unique identification number. To receive an identification number for a nonstandard format, the application must register the name of the format in the system atom table. Other applications that have the name of the format can then query the system atom table for the format's identification number. This method ensures that all applications use the same atom to identify a format.

DDESTRUCT Field - cbData

cbData ([ULONG](#))
Length of the data.

This is the length of data that occurs after the *offabData* parameter. If no data exists, this field should contain a zero (0).

DDESTRUCT Field - fsStatus

fsStatus ([USHORT](#))
Status of the data exchange.

DDE_FACK	Positive acknowledgement
DDE_FACKREQ	Acknowledgements are requested
DDE_FAPPSTATUS	A 1-byte field of bits that are reserved for application-specific returns.
DDE_FBUSY	Application is busy
DDE_FNODATA	No data transfer for advise
DDE_FRESPONSE	Response to WM_DDE_REQUEST
DDE_NOTPROCESSED	DDE message not understood

DDESTRUCT Field - usFormat

usFormat ([USHORT](#))
DDE data format.

This parameter can be set to one of the following values:

DDEFMT_TEXT	System-defined standard format for exchanging text data.
SZFMT_BITMAP	Specifies that the data is a bit map.
SZFMT_CPTTEXT	Specifies text whose format is defined by a CPTTEXT data structure. Applications can use this format to pass multiple-language strings without changing the conversation context.
SZFMT_DIF	Specifies that the data is in Data Image Format (DIF).
SZFMT_DSPBITMAP	Specifies that the data is a bit-map representation of a private data format.
SZFMT_DSPMETAFILE	Specifies that the data is a metafile representation of a private data format.
SZFMT_DSPMETAFILEPICT	Specifies that the data is a metafile picture representation of a private data format.
SZFMT_DSPTEXT	Specifies that the data is a text representation of a private data format.
SZFMT_LINK	Specifies that the data is in link-file format.
SZFMT_METAFILE	Specifies that the data is a metafile.
SZFMT_METAFILEPICT	Specifies that the data is a metafile picture defined by an MFP data structure.
SZFMT_OEMTEXT	Specifies that the data is in OEM Text format.
SZFMT_PALETTE	Specifies that the data is in palette format.
SZFMT_SYLK	Specifies that the data is in Synchronous Link format.
SZFMT_TEXT	Specifies that the data is an array of text characters. These characters can include new-line characters to indicate linebreaks. The zero-length character indicates the end of the text data.
SZFMT_TIFF	Specifies that the data is in Tag Image File Format (TIFF).

DDESTRUCT Field - offszItemName

offszItemName ([USHORT](#))
Offset to item name.

This is the offset to the item name from the start of this structure. Item name is a null (0x00) terminated string. If no item name exists, there must be a single null (0x00) character in this position. (That is, ItemName is ALWAYS a null terminated string.)

DDESTRUCT Field - offabData

offabData ([USHORT](#))
Offset to beginning of data.

This is the offset to the data, from the start of this structure. This field should be calculated regardless of the presence of data. If no data exists, **cbData** must be zero (0).

For compatibility reasons, this data should not contain embedded pointers. Offsets should be used instead.

DELETENOTIFY

Structure that contains information about the application page that is being deleted from a notebook.

```
typedef struct _DELETENOTIFY {
    HWND      hwndBook;      /* Notebook window handle. */
    HWND      hwndPage;      /* Application page window handle. */
    ULONG     ulAppPageData; /* Application-specified page data. */
    HBITMAP    hbmTab;        /* Application-specified tab bit map. */
} DELETENOTIFY;

typedef DELETENOTIFY *PDELETENOTIFY;
```

DELETENOTIFY Field - hwndBook

hwndBook ([HWND](#))
Notebook window handle.

DELETENOTIFY Field - hwndPage

hwndPage ([HWND](#))
Application page window handle.

DELETENOTIFY Field - ulAppPageData

ulAppPageData ([ULONG](#))
Application-specified page data.

DELETENOTIFY Field - hbmTab

hbmTab ([HBITMAP](#))
Application-specified tab bit map.

DESKTOP

Desktop background state structure.

```
typedef struct _DESKTOP {
    ULONG      cbSize;          /* Length of structure. */
    HBITMAP    hbm;            /* Bit-map handle of desktop background. */
    LONG       x;              /* X desktop coordinate of the origin of the bit map. */
    LONG       y;              /* Y desktop coordinate of the origin of the bit map. */
    ULONG      fl;             /* Desktop background state indicators or setting options. */
    LONG       lTileCount;     /* Number of images of the bit map to be tiled. */
    CHAR       szFile[260];    /* Zero-terminated name of the file containing the bit map. */
} DESKTOP;

typedef DESKTOP *PDESKTOP;
```

DESKTOP Field - cbSize

cbSize (ULONG)
Length of structure.

DESKTOP Field - hbm

hbm (HBITMAP)
Bit-map handle of desktop background.

DESKTOP Field - x

x (LONG)
X desktop coordinate of the origin of the bit map.

DESKTOP Field - y

y (LONG)
Y desktop coordinate of the origin of the bit map.

DESKTOP Field - fl

fl (ULONG)
Desktop background state indicators or setting options.

SDT_CENTER	The desktop background bit map is, or is to be, centered on the screen. If this option is specified, then the values of the <i>x</i> the <i>y</i> parameters are inapplicable.
SDT_DESTROY	Any existing desktop background bit map is to be destroyed. The setting of this option is not returned on the WinQueryDesktopBkgnd function.
SDT_LOADFILE	For the WinSetDesktopBkgnd function the bit map is to be loaded from the filename specified. If the SDT_NOBKGND flag is also set then the bit map is loaded but the background is not set. Tiling and scaling may be performed at load time or later when setting the bit map.
SDT_NOBKGND	There is no desktop background bit map, that is the desktop background i a solid color. For the WinQueryDesktopBkgnd function the existing background is to be left unmodified unless SDT_DESTROY is also specified.
SDT_PATTERN	The bit map represents a fill pattern.
SDT_RETAIN	The <i>szFile</i> is, or is to be, remembered for use when the system is started.
SDT_SCALE	The bit map is, or is to be, scaled to fill the desktop. If this option is specified, then the values of the <i>x</i> and <i>y</i> parameters are inapplicable.
SDT_TILE	The bit map is, or is to be, tiled to fill the desktop.

DESKTOP Field - ITileCount

ITileCount ([LONG](#))

Number of images of the bit map to be tiled.

The tile count is the number of images to be drawn in the vertical and horizontal direction when tiling the desktop background.

DESKTOP Field - szFile[260]

szFile[260] ([CHAR](#))

Zero-terminated name of the file containing the bit map.

DEVOPENSTRUC

Open-device data structure.

```
typedef struct _DEVOPENSTRUC {
    PSZ          pszLogAddress;    /* Logical address. */
    PSZ          pszDriverName;    /* Driver name. */
    PDRIVDATA    pdriv;            /* Driver data. */
    PSZ          pszDataType;      /* Data type. */
    PSZ          pszComment;       /* Comment. */
    PSZ          pszQueueProcName; /* Queue-processor name. */
    PSZ          pszQueueProcParams; /* Queue-processor parameters. */
    PSZ          pszSpoolerParams; /* Spooler parameters. */
    PSZ          pszNetworkParams; /* Network parameters. */
} DEVOPENSTRUC;
```

```
typedef DEVOPENSTRUC *PDEVOPENSTRUC;
```

DEVOPENSTRUC Field - pszLogAddress

pszLogAddress (PSZ)

Logical address.

This is required for an OD_DIRECT device being opened with DevOpenDC; it is the logical device address, such as "LPT1" on OS/2. Some drivers may accept a file name for this parameter or even a named pipe.

Where output is to be queued (for an OD_QUEUED device), this is the name of the queue for the output device. The queue name can be a UNC name.

Note: This parameter can be a port name for a printer device context.

DEVOPENSTRUC Field - pszDriverName

pszDriverName (PSZ)

Driver name.

Character string identifying the printer driver, for example, LASERJET. The Default Device Driver field of the [PRQINFO3](#) structure, associated with the required print queue, gives the driver and device name, separated by a period, for example LASERJET.HP LaserJet IIID. It can contain only the name up to the period, for example LASERJET.

DEVOPENSTRUC Field - pdriv

pdriv (PDRIVDATA)

Driver data.

Data that is to be passed directly to the PM device driver. Whether any of this is required depends upon the device driver.

For printer device context, this is a pointer to the job properties data.

DEVOPENSTRUC Field - pszDataType

pszDataType (PSZ)

Data type.

For an OD_QUEUED or OD_DIRECT device, this parameter defines the type of data that is to be queued as follows:

PM_Q_STD	Standard format
PM_Q_RAW	Raw format

Note that a device driver can define other data types.

For OD_QUEUED or OD_DIRECT device types, the default is supplied by the device driver if *pszDataType* is not specified. For any other device type, *pszDataType* is ignored.

DEVOPENSTRUC Field - pszComment

pszComment (PSZ)

Comment.

Optional character string that the printer object displays to the user in a job settings notebook. It is recommended that the application include its own name in this comment string.

Note: The job title text is derived from the document name passed to [DevEscape](#) (DEVESC_STARTDOC).

DEVOPENSTRUC Field - pszQueueProcName

pszQueueProcName (PSZ)

Queue-processor name.

This is the name of the queue processor, for queued output, and is usually the default.

DEVOPENSTRUC Field - pszQueueProcParams

pszQueueProcParams (PSZ)

Queue-processor parameters.

Queue processor parameters (optional). They can include information such as the number of copies you want to print and the size of the output area on the printed page.

The first parameter (*COP*) is used for all spool-file formats. The remaining parameters are valid for PM_Q_STD spool files only. Because PM_Q_STD data are used mainly for *graphic* data, these parameters are described in relation to the printing of picture files.

The PMPRINT/PMPLLOT queue-processor parameters are separated by spaces and are:

COP=*n*

The *COP* parameter specifies the number of copies of the spool file that you want printed. The value of *n* must be an integer in the range of 1 through 999.

The default is *COP=1*.

ARE=C | *w,h,l,t*

The *ARE* parameter determines the size and position of the output area. This is the area of the physical page to which printing is restricted.

The default value of *ARE=C* means that the output area is the whole page. Note, however, that the printer cannot print outside its own device clip limits.

To size and position the output area at a specific point on the page, use *ARE=w,h,l,t*, where:

w, *h*

are the width and height of the desired output area.

l, t are the offsets of the upper-left corner of the output area from the left (l) and from the top (t) of the maximum output area.

These four values must be given as percentages of the maximum output dimensions. The maximum output area is the area within the device clip limits.

$FIT=S \mid l, t$

The FIT parameter determines which part of the picture is to be printed. You can request the whole of the picture, scaled to fit the output area; or you can position the picture (actual size) anywhere within the output area. This could mean that the picture is clipped at the boundaries of the output area.

The default value of $FIT=S$ causes the output to be scaled until the larger of the height or width just fits within the defined output area. The aspect ratio of the picture is maintained.

To print the picture in actual size, use $FIT=l, t$, where l, t are the coordinates of the point in the picture that you want positioned at the center of the output area: l is measured from the left edge of the picture; and t is measured from the top edge. The coordinates must be given as percentages of the actual dimensions of the picture.

$XF=0 \mid 1$

The XF parameter enables you to override the picture-positioning and clipping instructions that are provided by the ARE and FIT parameters, including their defaults.

The default value of $XF=1$ allows the appearance of the output to be determined by the settings of the ARE and FIT parameters.

A value of $XF=0$ yields output as specified in the picture file. For example, applications that use many different forms can define different positions on each form for their output.

$COL=M \mid C$

The COL parameter enables you to specify color output if you have a color printer.

A value of $COL=M$ creates monochrome output (black foreground with no background color). This is supported by all devices.

A value of $COL=C$ creates color output. If you request color output on a monochrome device, the printer presentation driver tries to satisfy your request, which can cause problems because the only color available is black. For example, if the picture file specifies a red line on a blue background, both are drawn in black.

The default is $COL=M$ when you are addressing a monochrome printer and $COL=C$ when you are addressing a color printer.

$MAP=N \mid A$

The MAP parameter enables you to decide how the *neutral* colors (those that are not specified in the picture file) are printed.

The default value of $MAP=N$ yields a *normal* representation of the screen picture on a printed page, which means that the page background is white and the foreground is black.

A value of $MAP=A$ provides the reverse of the normal representation: the background is black and the foreground is white on the printed page.

$CDP=codepage$

The CDP parameter overrides the code page to be used for PM_Q_RAW print jobs. The print queue driver uses DEVESC_SETMODE to set the code page, but not all printer drivers support this device escape.

$XLT=0 \mid 1$

The XLT parameter can eliminate the translation component when printing a metafile if $XLT=1$.

When the resolution of the device is higher than that of the world coordinate space, a small translation of world coordinate point (0,0) occurs on the device to preserve the accuracy of the mapping from world to device coordinate units. For example, (0,0) becomes (1,1) if there are 3 pels to every world coordinate.

Normally, this is not noticeable, but it can be a problem with some devices. For example, in order to draw a complete row of 80 characters using a device font, a device may require the text to start at device coordinate position zero. Starting at a position other than zero may cause one or more characters at the end of the row to be clipped. In such cases, elimination of the translation is important and can be accomplished by specifying $XLT=1$.

The default is $XLT=0$.

DEVOPENSTRUC Field - pszSpoolerParams

pszSpoolerParams (PSZ)

Spooler parameters.

Spooler parameters (optional) are separated by spaces. They are used for scheduling print jobs and are as follows:

- The form names that identify the paper to be used, for example, *FORM=A4,A5,ENV*. The form names are optional; but if they are provided, the spooler is able to hold off printing the jobs until the required form is installed in the printer. If the form name is not provided, the spooler attempts to print the job. The printer driver recognizes that there is a forms problem and displays a FORMS MISMATCH message box.
- Priority of the print job, for example, *PRTY=60*. The priority is specified as an integer in the range 1 through 99; 99 is the highest. The default priority value is 50. The application can use the spooler priority parameter to prioritize its own jobs; however, it is not good practice for an application always to use priority 99 in an attempt to get its jobs printed first.

DEVOPENSTRUC Field - pszNetworkParams

pszNetworkParams (PSZ)

Network parameters.

Optional parameter that can be used to specify network options; for example, *USER=JOESMITH*.

DLGTEMPLATE

Dialog-template structure.

```
typedef struct _DLGTEMPLATE {
    USHORT    cbTemplate;      /* Length of template. */
    USHORT    type;            /* Template format type. */
    USHORT    codepage;        /* Code page. */
    USHORT    offadlgti;        /* Offset to dialog items. */
    USHORT    fsTemplateStatus; /* Template status. */
    USHORT    iItemFocus;       /* Index of item to receive focus initially. */
    USHORT    coffPresParams;    /* Count of presentation-parameter offsets. */
    DLGTITEM  adlgti[1];        /* Start of dialog items. */
} DLGTEMPLATE;

typedef DLGTEMPLATE *PDLGTEMPLATE;
```

DLGTEMPLATE Field - cbTemplate

cbTemplate (USHORT)

Length of template.

DLGTEMPLATE Field - type

type (USHORT)
Template format type.

DLGTEMPLATE Field - codepage

codepage (USHORT)
Code page.

DLGTEMPLATE Field - offadlgti

offadlgti (USHORT)
Offset to dialog items.

DLGTEMPLATE Field - fsTemplateStatus

fsTemplateStatus (USHORT)
Template status.

DLGTEMPLATE Field - iltemFocus

iltemFocus (USHORT)
Index of item to receive focus initially.

DLGTEMPLATE Field - coffPresParams

coffPresParams (USHORT)
Count of presentation-parameter offsets.

DLGTEMPLATE Field - adlgti[1]

adlgti[1] ([DLGITEM](#))
Start of dialog items.

DLGITEM

Dialog-item structure.

```
typedef struct _DLGITEM {
    USHORT    fsItemStatus; /* Status. */
    USHORT    cChildren;    /* Count of children to this dialog item. */
    USHORT    cchClassName; /* Length of class name. */
    USHORT    offClassName; /* Offset to class name. */
    USHORT    cchTextLen;   /* Length of text. */
    USHORT    offText;      /* Offset to text. */
    ULONG     flStyle;      /* Dialog item window style. */
    SHORT     x;            /* X-coordinate of origin of dialog-item window. */
    SHORT     y;            /* Y-coordinate of origin of dialog-item window. */
    SHORT     cx;           /* Dialog-item window width. */
    SHORT     cy;           /* Dialog-item window height. */
    USHORT    id;           /* Identity. */
    USHORT    offPresParams; /* Reserved. */
    USHORT    offCtlData;   /* Offset to control data. */
} DLGITEM;

typedef DLGITEM *PDLGITEM;
```

DLGITEM Field - fsItemStatus

fsItemStatus ([USHORT](#))
Status.

DLGITEM Field - cChildren

cChildren ([USHORT](#))
Count of children to this dialog item.

DLGITEM Field - cchClassName

cchClassLen (USHORT)
Length of class name.

If zero, *offClassName* contains the hexadecimal equivalent of a preregistered class name.

DLGTITEM Field - offClassName

offClassName (USHORT)
Offset to class name.

If *cchClassLen* is nonzero, this is the offset to a null-terminated ASCII string that contains the classname. If *cchClassLen* is zero, this is of the form 0xhhhh, where hhhh is the hexadecimal equivalent of the preregistered class name.

DLGTITEM Field - cchTextLen

cchTextLen (USHORT)
Length of text.

DLGTITEM Field - offText

offText (USHORT)
Offset to text.

DLGTITEM Field - flStyle

flStyle (ULONG)
Dialog item window style.

The high-order 16 bits are the standard WS_* style bits. The low-order 16 bits are available for class-specific use.

DLGTITEM Field - x

x (SHORT)
X-coordinate of origin of dialog-item window.

DLGTITEM Field - y

y (SHORT)
Y-coordinate of origin of dialog-item window.

DLGTITEM Field - cx

cx (SHORT)
Dialog-item window width.

DLGTITEM Field - cy

cy (SHORT)
Dialog-item window height.

DLGTITEM Field - id

id (USHORT)
Identity.

DLGTITEM Field - offPresParams

offPresParams (USHORT)
Reserved.

DLGTITEM Field - offCtlData

offCtlData (USHORT)
Offset to control data.

DRAGIMAGE

Dragged-object-image structure which describes the images that are to be drawn under the direct-manipulation pointer for the duration of a drag operation.

```
typedef struct _DRAGIMAGE {  
    USHORT      cb;           /* Size, in bytes, of the DRAGIMAGE structure. */  
    USHORT      cptl;         /* The number of points in the point array if fl is specified as DRG_POLYGON. */  
    LHANDLE      hImage;       /* Handle representing the image to display. */  
    SIZEL        sizlStretch;  /* Dimensions for stretching when fl is specified as DRG_STRETCH. */  
    ULONG        fl;          /* Flags. */  
    SHORT        cxOffset;     /* X-offset from the pointer hot spot to the origin of the image. */  
    SHORT        cyOffset;     /* Y-offset from the pointer hot spot to the origin of the image. */  
} DRAGIMAGE;  
  
typedef DRAGIMAGE *PDRAGIMAGE;
```

DRAGIMAGE Field - cb

cb ([USHORT](#))
Size, in bytes, of the DRAGIMAGE structure.

DRAGIMAGE Field - cptl

cptl ([USHORT](#))
The number of points in the point array if *fl* is specified as DRG_POLYGON.

DRAGIMAGE Field - hImage

hImage ([LHANDLE](#))
Handle representing the image to display.

The type is determined by *fl*.

DRAGIMAGE Field - sizlStretch

sizlStretch ([SIZEL](#))
Dimensions for stretching when *fl* is specified as DRG_STRETCH.

DRAGIMAGE Field - fl

fl ([ULONG](#))

Flags.

DRG_ICON

hImage is an [HPOINTER](#).

DRG_BITMAP

hImage is an [HBITMAP](#).

DRG_POLYGON

hImage is a pointer to an array of points that will be connected with GpiPolyLine to form a polygon. The first point of the array should be (0,0), and the other points should be placed relative to this position.

DRG_STRETCH

If DRG_ICON or DRG_BITMAP is specified, the image is expanded or compressed to the dimensions specified by *szl/Stretch*.

DRG_TRANSPARENT

If DRG_ICON is specified, an outline of the icon is generated and displayed instead of the original icon.

DRG_CLOSED

If DRG_POLYGON is specified, a closed polygon is formed by moving the current position to the last point in the array before calling GpiPolyLine.

DRAGIMAGE Field - cxOffset

cxOffset ([SHORT](#))

X-offset from the pointer hot spot to the origin of the image.

DRAGIMAGE Field - cyOffset

cyOffset ([SHORT](#))

Y-offset from the pointer hot spot to the origin of the image.

DRAGINFO

Drag-information structure.

```
typedef struct _DRAGINFO {
    ULONG      cbDraginfo; /* Structure size, in bytes. */
    USHORT     cbDragitem; /* Size, in bytes, of each DRAGITEM structure. */
    USHORT     usOperation; /* Modified drag operations. */
    HWND       hwndSource; /* Window handle of the source of the drag operation. */
    SHORT      xDrop; /* X-coordinate of drop point expressed in desktop coordinates. */
    SHORT      yDrop; /* Y-coordinate of drop point expressed in desktop coordinates. */
    USHORT     cditem; /* Count of DRAGITEM structures. */
    USHORT     usReserved; /* Reserved. */
}
```

```
} DRAGINFO;  
  
typedef DRAGINFO *PDRAGINFO;
```

DRAGINFO Field - cbDraginfo

cbDraginfo ([ULONG](#))

Structure size, in bytes.

The size includes the array of [DRAGITEM](#) structures.

DRAGINFO Field - cbDragitem

cbDragitem ([USHORT](#))

Size, in bytes, of each [DRAGITEM](#) structure.

DRAGINFO Field - usOperation

usOperation ([USHORT](#))

Modified drag operations.

An application can define its own modified drag operations for use when simulating a drop. These operations must have a value greater than DO_UNKNOWN. Possible values are described in the following list:

DO_DEFAULT	Execute the default drag operation. No modifier keys are pressed.
DO_COPY	Execute a copy operation. The Ctrl key is pressed.
DO_LINK	Execute a link operation. The Ctrl+Shift keys are pressed.
DO_MOVE	Execute a move operation. The Shift key is pressed.
DO_CREATE	Execute a create operation. (A template is being dropped.)
DO_NEW	Execute a create another operation. This value should be defined as DO_UNKNOWN+3 if it is not recognized in the current level of the toolkit.
DO_UNKNOWN	An undefined combination of modifier keys is pressed.

DRAGINFO Field - hwndSource

hwndSource ([HWND](#))

Window handle of the source of the drag operation.

DRAGINFO Field - xDrop

xDrop ([SHORT](#))
X-coordinate of drop point expressed in desktop coordinates.

DRAGINFO Field - yDrop

yDrop ([SHORT](#))
Y-coordinate of drop point expressed in desktop coordinates.

DRAGINFO Field - cditem

cditem ([USHORT](#))
Count of [DRAGITEM](#) structures.

DRAGINFO Field - usReserved

usReserved ([USHORT](#))
Reserved.

DRAGITEM

Drag-object structure.

```
typedef struct _DRAGITEM {
    HWND         hwndItem;          /* Window handle of the source of the drag operation. */
    ULONG        ulItemID;          /* Information used by the source to identify the object being dragged. */
    HSTR         hstrType;          /* String handle of the object type. */
    HSTR         hstrRMF;           /* String handle of the rendering mechanism and format. */
    HSTR         hstrContainerName; /* String handle of the name of the container holding the source object. */
    HSTR         hstrSourceName;    /* String handle of the name of the source object. */
    HSTR         hstrTargetName;    /* String handle of the suggested name of the object at the target. */
    SHORT        cxOffset;         /* X-offset from the pointer hot spot to the origin of the image that represe
    SHORT        cyOffset;         /* Y-offset from the pointer hot spot to the origin of the image that represe
    USHORT       fsControl;        /* Source-object control flags. */
    USHORT       fsSupportedOps;   /* Direct manipulation operations supported by the source object. */
}
```



```
} DRAGITEM;  
  
typedef DRAGITEM *PDRAGITEM;
```

DRAGITEM Field - hwndItem

hwndItem ([HWND](#))

Window handle of the source of the drag operation.

DRAGITEM Field - ullItemID

ullItemID ([ULONG](#))

Information used by the source to identify the object being dragged.

DRAGITEM Field - hstrType

hstrType ([HSTR](#))

String handle of the object type.

The string handle must be created using the [DrgAddStrHandle](#) function. The string is of the form:

`type[, type...]`

The first type in the list must be the true type of the object. The following types are used by the OS/2* shell:

DRT_ASM	Assembler code
DRT_BASIC	BASIC code
DRT_BINDATA	Binary data
DRT_BITMAP	Bit map
DRT_C	C code
DRT_COBOL	COBOL code
DRT_DLL	Dynamic link library
DRT_DOSCMD	DOS command file
DRT_EXE	Executable file
DRT_FONT	Font
DRT_FORTRAN	FORTTRAN code
DRT_ICON	

DRT_LIB	Icon
DRT_METAFILE	Library
DRT_OS2CMD	Metafile
DRT_PASCAL	OS/2 command file
DRT_RESOURCE	Pascal code
DRT_TEXT	Resource file
DRT_UNKNOWN	Text
	Unknown type.

DRAGITEM Field - hstrRMF

hstrRMF ([HSTR](#))

String handle of the rendering mechanism and format.

The string handle must be created using the [DrgAddStrHandle](#) function. The string is of the form:

```
mechfmt [, mechfmt . . . ]
```

where mechfmt can be in either of the following formats:

- <mechanism(1),format(1)>
- (mechanism(1)[, mechanism(n)...]), (format(1)[,format(n)...])

The first mechanism/format pair must be the native rendering mechanism and format of the object.

Valid mechanisms are:

"DRM_DDE"	Dynamic data exchange
"DRM_OBJECT"	Item being dragged is a Workplace Shell object
"DRM_OS2FILE"	OS/2 file
"DRM_PRINT"	Object can be printed using direct manipulation.

Valid formats are:

"DRF_BITMAP"	OS/2 bit map
"DRF_DIB"	DIB
"DRF_DIF"	DIF
"DRF_DSPBITMAP"	Stream of bit-map bits
"DRF_METAFILE"	Metafile
"DRF_OEMTEXT"	OEM text
"DRF_OWNERDISPLAY"	Bit stream
"DRF_PTRPICT"	Printer picture
"DRF_RTF"	Rich text
"DRF_SYLK"	SYLK

"DRF_TEXT"	Null-terminated string
"DRF_TIFF"	TIFF
"DRF_UNKNOWN"	Unknown format.

DRAGITEM Field - hstrContainerName

hstrContainerName ([HSTR](#))

String handle of the name of the container holding the source object.

The string handle must be created using the [DrgAddStrHandle](#) function.

DRAGITEM Field - hstrSourceName

hstrSourceName ([HSTR](#))

String handle of the name of the source object.

The string handle must be created using the [DrgAddStrHandle](#) function.

DRAGITEM Field - hstrTargetName

hstrTargetName ([HSTR](#))

String handle of the suggested name of the object at the target.

It is the responsibility of the source of the drag operation to create this string handle before calling [DrgDrag](#).

DRAGITEM Field - cxOffset

cxOffset ([SHORT](#))

X-offset from the pointer hot spot to the origin of the image that represents this object.

This value is copied from *cxOffset* in the [DRAGIMAGE](#) structure by [DrgDrag](#).

DRAGITEM Field - cyOffset

cyOffset (SHORT)

Y-offset from the pointer hot spot to the origin of the image that represents this object.

This value is copied from *cyOffset* in the DRAGIMAGE structure by DrgDrag.

DRAGITEM Field - fsControl

fsControl (USHORT)

Source-object control flags.

DC_OPEN	Object is open
DC_REF	Reference to another object
DC_GROUP	Group of objects
DC_CONTAINER	Container of other objects
DC_PREPARE	Source requires a DM_RENDERPREPARE message before it establishes a data transfer conversation
DC_REMOVEABLEMEDIA	Object is on removable media, or object cannot be recovered after a move operation.

DRAGITEM Field - fsSupportedOps

fsSupportedOps (USHORT)

Direct manipulation operations supported by the source object.

DO_COPYABLE	Source supports DO_COPY
DO_LINKABLE	Source supports DO_LINK
DO_MOVEABLE	Source supports DO_MOVE.

DRAGTRANSFER

Drag-conversation structure.

```
typedef struct _DRAGTRANSFER {
    ULONG      cb;                /* Size, in bytes, of the structure. */
    HWND       hwndClient;        /* Handle of the client window. */
    PDRAITEM   pditem;           /* Pointer to the DRAGITEM structure that is to be rendered. */
    HSTR       hstrSelectedRMF;   /* String handle for the selected rendering mechanism and format for the tr
    HSTR       hstrRenderToName;  /* String handle representing the name where the source places, and the tar
    ULONG      ulTargetInfo;      /* Reserved. */
    USHORT     usOperation;       /* The operation. */
    USHORT     fsReply;          /* Reply flags. */
} DRAGTRANSFER;

typedef DRAGTRANSFER *PDRAFTERTRANSFER;
```

DRAGTRANSFER Field - cb

cb ([ULONG](#))

Size, in bytes, of the structure.

DRAGTRANSFER Field - hwndClient

hwndClient ([HWND](#))

Handle of the client window.

This can be the target window or a window that represents an object in a container that was dropped on.

DRAGTRANSFER Field - pditem

pditem ([PDRAGITEM](#))

Pointer to the [DRAGITEM](#) structure that is to be rendered.

This structure must exist within the [DRAGINFO](#) structure that was passed in the [DM_DROP](#) message.

DRAGTRANSFER Field - hstrSelectedRMF

hstrSelectedRMF ([HSTR](#))

String handle for the selected rendering mechanism and format for the transfer operation.

This handle must be created using [DrgAddStrHandle](#). The target is responsible for deleting this handle when the conversation is complete. The string is in the format: [<mechanism,format>](#).

DRAGTRANSFER Field - hstrRenderToName

hstrRenderToName ([HSTR](#))

String handle representing the name where the source places, and the target finds, the data that is rendered.

The target is responsible for deleting this string handle when the conversation terminates. The contents of this field vary according to the rendering mechanism. See the *hstrRMF* field in [DRAGITEM](#).

OS/2 File

The string handle represents the fully qualified name of the file where the rendering will be placed.

DDE
Print

This field is not used.
This field is not used.

DRAGTRANSFER Field - ulTargetInfo

ulTargetInfo ([ULONG](#))
Reserved.

Reserved for use by the target. The target can use this field for information about the object and rendering operation.

DRAGTRANSFER Field - usOperation

usOperation ([USHORT](#))
The operation.

Values are:

DO_COPY

Execute a copy operation.

DO_LINK

Execute a link operation.

DO_MOVE

Execute a move operation.

DO_CREATE

Execute a create operation.

DO_NEW

Execute a create another operation. This value should be defined as DO_UNKNOWN+3 if it is not recognized in the current level of the toolkit.

Other

Execute an application-defined operation.

DRAGTRANSFER Field - fsReply

fsReply ([USHORT](#))
Reply flags.

Reply flags for the message. These flags can be set as follows:

DMFL_NATIVERENDER

The source does not support rendering for this object. A source should not set this flag unless it provides sufficient information at the time of the drop for the target to perform the rendering operation. The target must send [DM_ENDCONVERSATION](#) to the source after carrying out the rendering operation or when it elects not to do a native rendering.

DMFL_RENDERERRETRY

The source supports rendering for the object but does not support the selected rendering mechanism and format. The target can try another mechanism and format by sending another [DM_RENDER](#) message. If the target does not retry, it must send a [DM_RENDERERCOMPLETE](#) message to the source. This flag is set in conjunction with the DMFL_NATIVERENDER flag.

DRIVDATA

Driver-data structure.

```
typedef struct _DRIVDATA {  
    LONG    cb;           /* Length. */  
    LONG    lVersion;     /* Version. */  
    CHAR    szDeviceName[32]; /* Device name. */  
    CHAR    abGeneralData[1]; /* General data. */  
} DRIVDATA;  
  
typedef DRIVDATA *PDRIVDATA;
```

DRIVDATA Field - cb

cb ([LONG](#))

Length.

The length of the structure.

DRIVDATA Field - lVersion

lVersion ([LONG](#))

Version.

The version number of the data. Version numbers are defined by particular PM device drivers.

DRIVDATA Field - szDeviceName[32]

szDeviceName[32] ([CHAR](#))

Device name.

A string in a 32-byte field identifying the particular device (model number, and so on). Again, valid values are defined by PM device drivers.

DRIVDATA Field - abGeneralData[1]

abGeneralData[1] ([CHAR](#))

General data.

Data as defined by the Presentation Manager device driver.

The data type of this field is defined by the Presentation Manager device driver. It does not contain pointers, as these are not necessarily valid when passed to the device driver.

ENTRYFDATA

Entry-field control data structure.

```
typedef struct _ENTRYFDATA {
    USHORT      cb;          /* Length of control data in bytes. */
    USHORT      cchEditLimit; /* Edit limit. */
    USHORT      ichMinSel;    /* Minimum selection. */
    USHORT      ichMaxSel;    /* Maximum selection. */
} ENTRYFDATA;

typedef ENTRYFDATA *PENTRYFDATA;
```

ENTRYFDATA Field - cb

cb (USHORT)

Length of control data in bytes.

The length of the control data for an entry field control.

ENTRYFDATA Field - cchEditLimit

cchEditLimit (USHORT)

Edit limit.

This is the maximum number of characters that can be entered into the entry field control.

If the operator tries to enter more text into an entry field control than is specified by the text limit set by the [EM_SETTEXTLIMIT](#) message, the entry field control indicates the error by sounding the alarm and does not accept the characters.

ENTRYFDATA Field - ichMinSel

ichMinSel (USHORT)

Minimum selection.

ENTRYFDATA Field - ichMaxSel

ichMaxSel (USHORT)

Maximum selection.

The *ichMinSel* and *ichMaxSel* parameters identify the current selection within the entry field control. Characters within the text with byte offsets less than the *ichMaxSel* parameter and greater than or equal to the *ichMinSel* parameter are the current selection. The cursor is positioned immediately before the character identified by the *ichMaxSel* parameter.

If the *ichMinSel* parameter is equal to the *ichMaxSel* parameter, the current selection becomes the insertion point.

If the *ichMinSel* parameter is equal to 0 and the *ichMaxSel* is greater than or equal to text limit set by the [EM_SETTEXTLIMIT](#) message, the entire text is selected.

ERRINFO

Error-information structure.

```
typedef struct _ERRINFO {
    ULONG      cbFixedErrInfo; /* Length of fixed data to this structure. */
    ERRORID    idError;        /* Error identity. */
    ULONG      cDetailLevel;   /* Number of levels of detail. */
    ULONG      offaoffszMsg;    /* Offset to the array of message offsets. */
    ULONG      offBinaryData;   /* Offset to the binary data. */
} ERRINFO;

typedef ERRINFO *PERRINFO;
```

ERRINFO Field - cbFixedErrInfo

cbFixedErrInfo (ULONG)

Length of fixed data to this structure.

ERRINFO Field - idError

idError (ERRORID)

Error identity.

This is identical to the value returned by `WinGetLastError`.

ERRINFO Field - cDetailLevel

cDetailLevel (ULONG)
Number of levels of detail.

This is the number of entries in the array of words pointed to by the following field. One level of detail is provided.

ERRINFO Field - offaoffszMsg

offaoffszMsg (ULONG)
Offset to the array of message offsets.

This is an offset to an array of 16-bit offsets to null-terminated strings. Each string is a printable message that offers varying levels of information. The first level is the least amount of detail, and the remaining levels offer more and more detail.

The first level of detail is always an error message string, in the following format:

```
xxxxnnnns
where  xxx  is the product identifier
      nnnn  is the message number
      s     is the message severity letter
           W = warning
           E = error
           S = severe error
           U = unrecoverable
```

ERRINFO Field - offBinaryData

offBinaryData (ULONG)
Offset to the binary data.

This can contain additional information relating to the error.

ERRORID

Error identity.

```
typedef ULONG ERRORID;
```

ESCMODE

Structure for setting printer mode. See GreEscape DEVESC_SETMODE This data structure is a more general version of the of the

ESCSETMODE data structure.

```
typedef struct _ESCMODE {
    ULONG     mode;          /* Mode. */
    BYTE      modedata[1];  /* Mode data. */
} ESCMODE;

typedef ESCMODE *PESCMODE;
```

ESCMODE Field - mode

mode (**ULONG**)
Mode.

ESCMODE Field - modedata[1]

modedata[1] (**BYTE**)
Mode data.

ESCSETMODE

Structure for setting printer mode. See [DevEscape](#)

```
typedef struct _ESCSETMODE {
    ULONG     mode;          /* Mode to be set. */
    USHORT    codepage;     /* Code page. */
} ESCSETMODE;

typedef ESCSETMODE *PESCSETMODE;
```

This data structure is a specific-case version of the **ESCMODE** data structure, used to set the code page of a printer.

ESCSETMODE Field - mode

mode (**ULONG**)
Mode to be set.

0 Set mode to specified code page. Any font can be used.

ESCSETMODE Field - codepage

codepage (USHORT)
Code page.

If zero is specified for the code page, the printer is set to the hardware default.

FACENAMEDESC

Face-name description structure. See GpiQueryFaceString.

```
typedef struct _FACENAMEDESC {
    USHORT    usSize;           /* Length of structure. */
    USHORT    usWeightClass;    /* Weight class. */
    USHORT    usWidthClass;     /* Width class. */
    USHORT    usReserved;       /* Reserved. */
    ULONG     flOptions;         /* Other characteristics of the font. */
} FACENAMEDESC;

typedef FACENAMEDESC *PFACENAMEDESC;
```

FACENAMEDESC Field - usSize

usSize (USHORT)
Length of structure.

FACENAMEDESC Field - usWeightClass

usWeightClass (USHORT)
Weight class.

Indicates the visual weight (thickness of strokes) of the characters in the font:

FWEIGHT_DONT_CARE	Any font weight satisfies the request.
FWEIGHT_ULTRA_LIGHT	Ultra-light.
FWEIGHT_EXTRA_LIGHT	Extra-light.
FWEIGHT_LIGHT	Light.
FWEIGHT_SEMI_LIGHT	Semi-light.
FWEIGHT_NORMAL	Medium (normal) weight.
FWEIGHT_SEMI_BOLD	Semi-bold.
FWEIGHT_BOLD	Bold.
FWEIGHT_EXTRA_BOLD	Extra-bold.

FWEIGHT_ULTRA_BOLD

Ultra-bold.

FACENAMEDESC Field - usWidthClass

usWidthClass (USHORT)
Width class.

Indicates the relative aspect ratio of the characters of the font in relation to the normal aspect ratio for this type of font:

FWIDTH_DONT_CARE	Any font width satisfies the request.
FWIDTH_ULTRA_CONDENSED	Ultra-condensed (50% of normal).
FWIDTH_EXTRA_CONDENSED	Extra-condensed (62.5% of normal).
FWIDTH_CONDENSED	Condensed (75% of normal).
FWIDTH_SEMI_CONDENSED	Semi-condensed (87.5% of normal).
FWIDTH_NORMAL	Medium (normal).
FWIDTH_SEMI_EXPANDED	Semi-expanded (112.5% of normal).
FWIDTH_EXPANDED	Expanded (125% of normal).
FWIDTH_EXTRA_EXPANDED	Extra-expanded (150% of normal).
FWIDTH_ULTRA_EXPANDED	Ultra-expanded (200% of normal).

FACENAMEDESC Field - usReserved

usReserved (USHORT)
Reserved.

FACENAMEDESC Field - flOptions

flOptions (ULONG)
Other characteristics of the font.

FTYPE_ITALIC	Italic font required. If not specified, non-italic font required.
FTYPE_ITALIC_DONT_CARE	Italic and non-italic fonts can satisfy the request. If this option is specified, FTYPE_ITALIC is ignored.
FTYPE_OBLIQUE	Oblique font required. If not specified, non-oblique font required.

FTYPE_OBLIQUE_DONT_CARE	Oblique and non-oblique fonts can satisfy the request. If this option is specified, FTYPE_OBLIQUE is ignored.
FTYPE_ROUNDED	Rounded font required. If not specified, non-rounded font required.
FTYPE_ROUNDED_DONT_CARE	Rounded and non-rounded fonts can satisfy the request. If this option is specified, FTYPE_ROUNDED is ignored.

FATTRS

Font-attributes structure.

```
typedef struct _FATTRS {
    USHORT    usRecordLength;      /* Length of record. */
    USHORT    fsSelection;         /* Selection indicators. */
    LONG      lMatch;              /* Matched-font identity. */
    CHAR      szFacename[FACESIZE]; /* Typeface name. */
    USHORT    idRegistry;          /* Registry identifier. */
    USHORT    usCodePage;          /* Code page. */
    LONG      lMaxBaselineExt;      /* Maximum baseline extension. */
    LONG      lAveCharWidth;        /* Average character width. */
    USHORT    fsType;              /* Type indicators. */
    USHORT    fsFontUse;           /* Font-use indicators. */
} FATTRS;

typedef FATTRS *PFATTRS;
```

FATTRS Field - usRecordLength

usRecordLength (USHORT)
Length of record.

FATTRS Field - fsSelection

fsSelection (USHORT)
Selection indicators.

Flags causing the following features to be simulated by the system.

Note: If an italic flag is applied to a font that is itself defined as italic, the font is slanted further by italic simulation.

Underscore or strikeout lines are drawn using the appropriate attributes (for example, color) from the character bundle (see the [CHARBUNDLE](#) datatype), not the line bundle (see [LINEBUNDLE](#)). The width of the line, and the vertical position of the line in font space, are determined by the font. Horizontally, the line starts from a point in font space directly above or below the start point of each character, and extends to a point directly above or below the escapement point for that character.

For this purpose, the start and escapement points are those applicable to left-to-right or right-to-left character directions (see

GpiSetCharDirection in *Graphics Programming Interface Programming Reference*), even if the string is currently being drawn in a top-to-bottom or bottom-to-top direction.

For left-to-right or right-to-left directions, any white space generated by the character extra and character break extra attributes (see GpiSetCharExtra and GpiSetCharBreakExtra in *Graphics Programming Interface Programming Reference*), as well as increments provided by the vector of increments on GpiCharStringPos and GpiCharStringPosAt, are also underlined/overstruck, so that in these cases the line is continuous for the string.

FATTR_SEL_ITALIC	Generate <i>italic</i> font.
FATTR_SEL_UNDERSCORE	Generate <u>underscored</u> font.
FATTR_SEL_BOLD	Generate bold font. (Note that the resulting characters are wider than those in the original font.)
FATTR_SEL_STRIKEOUT	Generate font with overstruck characters.
FATTR_SEL_OUTLINE	Use an outline font with hollow characters. If this flag is not set, outline font characters are filled. Setting this flag normally gives better performance, and for sufficiently small characters (depending on device resolution) there may be little visual difference.

FATTRS Field - IMatch

IMatch ([LONG](#))
Matched-font identity.

FATTRS Field - szFacename[FACESIZE]

szFacename[FACESIZE] ([CHAR](#))
Typeface name.

The typeface name of the font, for example, Tms Rmn.

FATTRS Field - idRegistry

idRegistry ([USHORT](#))
Registry identifier.

Font registry identifier (zero if unknown).

FATTRS Field - usCodePage

usCodePage ([USHORT](#))
Code page.

If zero, the current Gpi code page (see GpiSetCp in *Graphics Programming Interface Programming Reference*) is used. A subsequent GpiSetCp function changes the code page used for this logical font.

FATTRS Field - IMaxBaselineExt

IMaxBaselineExt ([LONG](#))
Maximum baseline extension.

For raster fonts, this should be the height of the required font, in world coordinates.

For outline fonts, this should be zero.

FATTRS Field - IAveCharWidth

IAveCharWidth ([LONG](#))
Average character width.

For raster fonts, this should be the width of the required font, in world coordinates.

For outline fonts, this should be zero.

FATTRS Field - fsType

fsType ([USHORT](#))
Type indicators.

FATTR_TYPE_KERNING	Enable kerning (PostScript only).
FATTR_TYPE_MBCS	Font for mixed single- and double-byte code pages.
FATTR_TYPE_DBCS	Font for double-byte code pages.
FATTR_TYPE_ANTIALIASED	Antialiased font required. Only valid if supported by the device driver.

FATTRS Field - fsFontUse

fsFontUse ([USHORT](#))
Font-use indicators.

These flags indicate how the font is to be used. They affect presentation speed and font quality.

FATTR_FONTUSE_NOMIX

Text is not mixed with graphics and can be written without regard to any interaction with graphics objects.

FATTR_FONTUSE_OUTLINE

Select an outline (vector) font. The font characters can be used as part of a path definition. If this flag is not set, an outline font might or might not be selected. If an outline font is selected, however, character widths are rounded to an integral number of pels.

FATTR_FONTUSE_TRANSFORMABLE

Characters can be transformed (for example, scaled, rotated, or sheared).

FFDESCS

Font-file descriptor.

```
typedef CHAR FFDESCS[2][FACESIZE];
```

FIELDINFO

Structure that contains information about column data in the details view of the container control. The details view displays each FIELDINFO structure as a column of data that contains specific information about each container record. For example, one FIELDINFO structure, or column, might contain icons or bit maps that represent each container record. Another FIELDINFO structure might contain the date or time that each container record was created.

```
typedef struct _FIELDINFO {
    ULONG      cb;                /* Structure size. */
    ULONG      flData;            /* Data attributes. */
    ULONG      flTitle;           /* Column heading attributes. */
    PVOID      pTitleData;        /* Column heading data. */
    ULONG      offStruct;         /* Structure offset. */
    PVOID      pUserData;         /* Pointer to user data. */
    struct _FIELDINFO *pNextFieldInfo; /* Pointer to the next linked FIELDINFO data structure. */
    ULONG      cxWidth;           /* Column width. */
} FIELDINFO;

typedef FIELDINFO *PFIELDINFO;
```

FIELDINFO Field - cb

cb (ULONG)

Structure size.

The size (in bytes) of the FIELDINFO structure.

FIELDINFO Field - flData

flData (ULONG)

Data attributes.

Attributes of the data in a field.

- Specify one of the following for each column to choose the type of data that is displayed in each column:

CFA_BITMAPORICON

The column contains bit-map or icon data.

CFA_DATE

The data in the column is displayed in date format. National Language Support (NLS) is enabled for date format. Use the data structure described in [CDATE](#)

CFA_STRING

Character or text data is displayed in this column.

CFA_TIME

The data in the column is displayed in time format. National Language Support (NLS) is enabled for time format. Use the data structure described in [CTIME](#).

CFA_ULONG

Unsigned number data is displayed in this column. National Language Support (NLS) is enabled for number format.

- Specify any or all of the following column attributes:

CFA_FIREADONLY

Prevents text in a FIELDINFO data structure (text in a column) from being edited directly. This attribute applies only to columns for which the CFA_STRING attribute has been specified.

CFA_HORZSEPARATOR

A horizontal separator is provided beneath column headings.

CFA_INVISIBLE

Invisible container column. The default is visible.

CFA_OWNER

Ownerdraw is enabled for this container column.

CFA_SEPARATOR

A vertical separator is drawn after this column.

- Specify one of the following for each column to vertically position data in that column:

CFA_BOTTOM

Bottom-justifies field data.

CFA_TOP

Top-justifies field data.

CFA_VCENTER

Vertically centers field data. This is the default.

- Specify one of the following for each column to horizontally position data in that column. These attributes can be combined with the attributes used for vertical positioning of column data by using an OR operator (|).

CFA_CENTER

Horizontally centers field data.

CFA_LEFT

Left-justifies field data. This is the default.

CFA_RIGHT

Right-justifies field data.

FIELDINFO Field - flTitle

flTitle (ULONG)

Column heading attributes.

- Specify the following if icon or bit-map data is to be displayed in the column heading:

CFA_BITMAPORICON

The column heading contains icon or bit-map data. If CFA_BITMAPORICON is not specified, any data that is assigned to a column heading is assumed to be character or text data.

- Specify the following to prevent direct editing of a column heading:

CFA_FITTLEREADONLY

Prevents a column heading from being edited directly.

- Specify one of the following for each column heading to vertically position data in that column heading:

CFA_TOP

Top-justifies column headings.

CFA_BOTTOM

Bottom-justifies column headings.

CFA_VCENTER

Vertically centers column headings. This is the default.

- Specify one of the following for each column heading to horizontally position data in that column heading. These attributes can be combined with the attributes used for vertical positioning of column heading data by using an OR operator (|).

CFA_CENTER

Horizontally centers column headings.

CFA_LEFT

Left-justifies column headings. This is the default.

CFA_RIGHT

Right-justifies column headings.

FIELDINFO Field - pTitleData

pTitleData (PVOID)

Column heading data.

Column heading data, which can be a text string or an icon or bit map. The default is a text string. If the *flTitle* field is set to the CFA_BITMAPORICON attribute, this must be an icon or bit map.

FIELDINFO Field - offStruct

offStruct (ULONG)

Structure offset.

Offset from the beginning of a RECORDCORE structure to the data that is displayed in this column.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

FIELDINFO Field - pUserData

pUserData ([PVOID](#))
Pointer to user data.

FIELDINFO Field - pNextFieldInfo

pNextFieldInfo (struct [_FIELDINFO](#) *)
Pointer to the next linked FIELDINFO data structure.

FIELDINFO Field - cxWidth

cxWidth ([ULONG](#))
Column width.

Used to specify the width of a column. The default is an automatically sized column that is always the width of its widest element. If this field is set and the data is too wide, the data is truncated.

FIELDINFOINSERT

Structure that contains information about the [FIELDINFO](#) structure or structures that are being inserted into a container. This structure is used in the [CM_INSERTDETAILFIELDINFO](#) container message only.

```
typedef struct _FIELDINFOINSERT {  
    ULONG      cb;                /* Structure size. */  
    PFIELDINFO pFieldInfoOrder;   /* Column order. */  
    ULONG      fInvalidateFieldInfo; /* Update flag. */  
    ULONG      cFieldInfoInsert;   /* Number of columns. */  
} FIELDINFOINSERT;  
  
typedef FIELDINFOINSERT *PFIELDINFOINSERT;
```

FIELDINFOINSERT Field - cb

cb ([ULONG](#))
Structure size.

The size (in bytes) of the FIELDINFOINSERT structure.

FIELDINFOINSERT Field - pFieldInfoOrder

pFieldInfoOrder ([PFIELDINFO](#))
Column order.

Orders the [FIELDINFO](#) structure or structures relative to other [FIELDINFO](#) structures in the container. The values can be:

CMA_FIRST	Places a FIELDINFO structure, or list of FIELDINFO structures, at the front of the list of columns.
CMA_END	Places a FIELDINFO structure, or list of FIELDINFO structures, at the end of the list of columns.
Other	Pointer to a FIELDINFO structure that this structure, or list of structures, is to be inserted after.

FIELDINFOINSERT Field - flInvalidateFieldInfo

flInvalidateFieldInfo ([ULONG](#))
Update flag.

Flag that indicates an automatic display update after the [FIELDINFO](#) structures are inserted.

TRUE	The display is automatically updated after FIELDINFO structures are inserted.
FALSE	The application must send the CM_INVALIDATEDDETAILFIELDINFO message after the FIELDINFO structures are inserted.

FIELDINFOINSERT Field - cFieldInfoInsert

cFieldInfoInsert ([ULONG](#))
Number of columns.

The number of [FIELDINFO](#) structures to be inserted. The *cFieldInfoInsert* field value must be greater than 0.

FILEDLG

File-dialog structure.

```

typedef struct _FILEDLG {
    ULONG      cbSize;          /* Structure size. */
    ULONG      fl;              /* FDS_* flags. */
    ULONG      ulUser;          /* Used by the application. */
    LONG       lReturn;         /* Result code. */
    LONG       lSRC;            /* System return code. */
    PSZ        pszTitle;        /* Dialog title string. */
    PSZ        pszOKButton;     /* OK push button text. */
    PFNWP      pfnDlgProc;      /* Custom dialog procedure. */
    PSZ        pszIType;        /* Extended-attribute type filter. */
    PAPSZ      papszITypeList;  /* Pointer to a table of pointers to extended-attribute types. */
    PSZ        pszIDrive;       /* The initial drive. */
    PAPSZ      papszIDriveList; /* Pointer to a table of pointers to drives. */
    HMODULE     hMod;           /* Module for custom dialog resources. */
    CHAR       szFullFile[CCHMAXPATH]; /* Character array. */
    PAPSZ      papszFQFilename; /* Pointer to a table of pointers to fully-qualified file names. */
    ULONG      ulFQFCount;      /* Number of file names. */
    USHORT     usDlgID;         /* Custom dialog ID. */
    SHORT      x;               /* X-axis dialog position. */
    SHORT      y;               /* Y-axis dialog position. */
    SHORT      sEAType;         /* Selected extended-attribute type. */
} FILEDLG;

typedef FILEDLG *PFILEDLG;

```

FILEDLG Field - cbSize

cbSize (ULONG)
Structure size.

Size of the structure. This field allows future expansion of the structure and must be initialized with the size of the FILEDLG structure.

FILEDLG Field - fl

fl (ULONG)
FDS_* flags.

Several flags can be specified to alter the behavior of the dialog.

Note: The dialog must be either an "Open" or a "Save As" dialog. If neither the FDS_OPEN_DIALOG nor the FDS_SAVEAS_DIALOG flag is set, or if both are set, the dialog will return an error.

FDS_APPLYBUTTON

An Apply push button is added to the dialog. This is useful in a modeless dialog.

FDS_CENTER

The dialog is positioned in the center of its parent window, overriding any specified *x*, *y* position.

FDS_CUSTOM

A custom dialog template is used to create the dialog. The *hMod* and *usDlgID* fields must be initialized.

FDS_ENABLEFILELB

When this flag is set, the Files list box on a Save As dialog is enabled. When this flag is not set, the Files list box is not enabled for a Save As dialog. This is the default.

FDS_FILTERUNION

When this flag is set, the dialog uses the union of the string filter and the extended-attribute type filter when filtering files for the Files list box. When this flag is not set, the list box, by default, uses the intersection of the two.

FDS_HELPBUTTON

A Help push button of style (BS_HELP|BS_NOINTERFOCUS) with an ID of DID_HELP_PB is added to the dialog. When this push button is pressed, a [WM_HELP](#) message is sent to

FDS_INCLUDE_EAS

If this flag is set, the dialog will always query extended attribute information for files as it fills the Files list box. The

	default is to not query the information unless an extended attribute type filter has been selected.
FDS_MODELESS	When this flag is set, the dialog is modeless; WinFileDlg returns immediately after creating the dialog window and returns the window handle to the application. The application should treat the dialog as if it were created with WinLoadDlg . As in the modal (default) dialog case, the return value is found in the <i>IReturn</i> field of the FILEDLG structure passed to WinFileDlg .
FDS_MULTIPLESEL	When this flag is set, the Files list box for the dialog is a multiple selection list box. When this flag is not set, the default is a single-selection list box.
FDS_OPEN_DIALOG	The dialog is an "Open" dialog when this flag is set.
FDS_PRELOAD_VOLINFO	If this flag is set, the dialog will preload the volume information for the drives and will preset the current default directory for each drive. The default behavior is for the volume label to be blank and the initial directory will be the root directory for each drive.
FDS_SAVEAS_DIALOG	The dialog is a "Save As" dialog when this flag is set.

FILEDLG Field - ulUser

ulUser ([ULONG](#))

Used by the application.

This field can be used by an application that is subclassing the file dialog to store its own state information.

FILEDLG Field - IReturn

IReturn ([LONG](#))

Result code.

Result code from dialog dismissal. This field contains the ID of the push button pressed to dismiss the dialog, DID_OK or DID_CANCEL, unless the application supplies additional push buttons in its template. If an error occurs on dialog invocation, this field is set to zero.

FILEDLG Field - ISRC

ISRC ([LONG](#))

System return code.

This field contains an FDS_ERR return code. When a dialog fails, this field is used to tell the application the reason for the failure.

FILEDLG Field - pszTitle

pszTitle ([PSZ](#))

Dialog title string.

When this field is NULL, the dialog title defaults to the name of the dialog currently running.

FILEDLG Field - pszOKButton

pszOKButton ([PSZ](#))

OK push button text.

This string is used to set the text of the OK push button. The default text is OK.

FILEDLG Field - pfnDlgProc

pfnDlgProc ([PFNWP](#))

Custom dialog procedure.

NULL unless the caller is subclassing the file dialog. When non-NULL, it points to the dialog procedure of the application.

FILEDLG Field - pszIType

pszIType ([PSZ](#))

Extended-attribute type filter.

This field contains a pointer to the initial extended-attribute type filter that is applied to the initial dialog screen. This filter is not required to be in *papszITypeList*.

FILEDLG Field - papszITypeList

papszITypeList ([PAPSZ](#))

Pointer to a table of pointers to extended-attribute types.

Each pointer in the table points to a null-terminated string, and each string is an extended-attribute type. These types are sorted in ascending order in the Type drop-down box. The end of the table is marked by a null pointer. To specify an empty table, the application sets this field to NULL, or it specifies a table containing only a null pointer.

FILEDLG Field - pszIDrive

pszDrive (PSZ)

The initial drive.

This field contains a pointer to a string that specifies the initial drive applied to the initial dialog screen. This drive is not required to be in *papszDriveList*.

FILEDLG Field - papszDriveList

papszDriveList (PAPSZ)

Pointer to a table of pointers to drives.

Each pointer in the table points to a null-terminated string, and each string is a valid drive or network identifier. These drives and network IDs will be sorted in ascending order in the Drive drop-down box. The end of the table is marked by a null pointer. To specify an empty table, the application sets this field to NULL, or it specifies a table containing only a null pointer.

FILEDLG Field - hMod

hMod (HMODULE)

Module for custom dialog resources.

If FDS_CUSTOM is set, this is the **HMODULE** from which the custom file dialog template is loaded. NULLHANDLE causes the dialog resource to be pulled from the module of the current EXE.

FILEDLG Field - szFullFile[CCHMAXPATH]

szFullFile[CCHMAXPATH] (CHAR)

Character array.

An array of characters where CCHMAXPATH is a system-defined constant. On initialization, this field contains the initial fully-qualified path and file name. On completion, this field contains the selected fully-qualified path and file name. The simple file name can be replaced with a string filter, such as *.DAT. When the dialog is invoked, all drive and path information is stripped from the entry and moved to the corresponding fields in the dialog.

When a file name is specified, the Files list box is scrolled to the matching file name. When there is no exact match, the closest match is used.

When a string filter is specified, the dialog is initially refreshed using the results of this filter intersected with the results of *pszType*. After the dialog is initially shown, the string filter remains in the file name field until a file is selected, or the user overtypes the value.

When a file is selected, **szFullFile** is returned to the calling application and is set to the selected fully-qualified file name.

When more than one file is selected in a multiple file selection dialog, only the topmost selected file name is returned in this field.

FILEDLG Field - papszFQFilename

papszFQFilename ([PAPSZ](#))

Pointer to a table of pointers to fully-qualified file names.

Returned to multiple file selection dialogs when the user selects one or more files from the list box. If the user types the file name in the file name entry field, the file name will be in **szFullFile** and this pointer will be NULL. When one or more selections are made, the count of items in this array will be returned in *uIFQFCount*.

This table of pointers is storage allocated by the file dialog. When the application completes opening or saving all of the files specified, the application must call [WinFreeFileDialogList](#) to free the storage allocated by the file dialog.

FILEDLG Field - uIFQFCount

uIFQFCount ([ULONG](#))

Number of file names.

Number of file names selected in the dialog. In a single file selection dialog, this value is 1. In a multiple file selection dialog, this value will be the number of files selected by the user.

FILEDLG Field - usDlgID

usDlgID ([USHORT](#))

Custom dialog ID.

The ID of the dialog window. When FDS_CUSTOM is set, this field contains the ID of the resource containing the custom dialog template.

FILEDLG Field - x

x ([SHORT](#))

X-axis dialog position.

This, along with *y* and is used to position the dialog. It is updated in the structure if the user moves the dialog to a new position. If the FILEDLG structure is reused, the dialog appears in the position at which it was left each time it is invoked. The FDS_CENTER flag overrides this position and automatically centers the dialog in its parent.

FILEDLG Field - y

y ([SHORT](#))

Y-axis dialog position.

This, along with *x* and is used to position the dialog. It is updated in the structure if the user moves the dialog to a new position. If the FILEDLG structure is reused, the dialog appears in the position at which it was left each time it is invoked. The FDS_CENTER flag overrides this position and automatically centers the dialog in its parent.

FILEDLG Field - sEAType

sEAType (SHORT)

Selected extended-attribute type.

Returns a selected extended-attribute type to assign to the file name returned in **szFullFile**. This field is a zero-based offset into the *papsz/TypeList* and is returned only when the Save As dialog is used. A -1 value is returned when the Open dialog is used.

FIXED

Signed-integer fraction (16:16). This can be treated as a [LONG](#) where the value has been multiplied by 65 536.

```
typedef LONG FIXED;
```

FONTDLG

Font-dialog structure.

```
typedef struct _FONTDLG {
    ULONG      cbSize;          /* Structure size. */
    HPS        hpsScreen;      /* Screen presentation space. */
    HPS        hpsPrinter;     /* Printer presentation space. */
    PSZ        pszTitle;       /* Dialog title string. */
    PSZ        pszPreview;     /* Font-preview window string. */
    PSZ        pszPtSizeList;  /* Application-provided point size list. */
    PFNWP      pfnDlgProc;     /* Custom dialog procedure. */
    PSZ        pszFamilyname;  /* Family name buffer. */
    FIXED      fxPointSize;    /* Point size of the font. */
    ULONG      fl;             /* FNTS_* flags. */
    ULONG      flFlags;        /* FNTF_* flags. */
    ULONG      flType;         /* The selected type bits. */
    ULONG      flTypeMask;     /* Mask of type bits to use. */
    ULONG      flStyle;        /* Selected style bits. */
    ULONG      flStyleMask;    /* Mask of style bits to use. */
    LONG       clrFore;        /* Font foreground color. */
    LONG       clrBack;        /* Font background color. */
    ULONG      ulUser;         /* Application-defined. */
    LONG       lReturn;        /* Return value. */
    LONG       lSRC;           /* System return code. */
    LONG       lEmHeight;      /* Em height. */
    LONG       lXHeight;       /* X height. */
    LONG       lExternalLeading; /* External leading. */
    HMODULE     hMod;          /* Module for custom dialog resources. */
    FATTRS      fAttrs;        /* Font-attribute structure. */
    SHORT       sNominalPointSize; /* Font point size. */
    USHORT      usWeight;      /* Font weight. */
    USHORT      usWidth;       /* Font width. */
    SHORT       x;             /* The x-axis dialog position. */
    SHORT       y;             /* The y-axis dialog position. */
    USHORT      usDlgId;       /* Dialog ID. */
    USHORT      usFamilyBufLen; /* Buffersize. */
    USHORT      usReserved;    /* Reserved. */
} FONTDLG;
```

```
typedef FONTDLG *PFONTDLG;
```

FONTDLG Field - cbSize

cbSize ([ULONG](#))
Structure size.

This field allows for future expansion of the structure, and must be initialized with the size of the FONTDLG structure.

FONTDLG Field - hpsScreen

hpsScreen ([HPS](#))
Screen presentation space.

If not NULLHANDLE, the screen presentation space from which screen fonts are queried.

FONTDLG Field - hpsPrinter

hpsPrinter ([HPS](#))
Printer presentation space.

If not NULLHANDLE, the printer presentation space from which printer font are queried.

FONTDLG Field - pszTitle

pszTitle ([PSZ](#))
Dialog title string.

Application-provided dialog title. If NULL, it defaults to "Font".

FONTDLG Field - pszPreview

pszPreview ([PSZ](#))
Font-preview window string.

String to show in font-preview window. If NULL, it defaults to "abcdABCD".

Note: Care is necessary when choosing the string to put in this field. Using many different characters causes excess memory to be used by the font cache.

FONTDLG Field - pszPtSizeList

pszPtSizeList (PSZ)

Application-provided point size list.

String which contains a list of point sizes to be used as the default list for outline fonts in the point-size drop-down area. Point sizes are separated by spaces. If NULL, the point size drop down defaults to 8, 10, 12, 14, 18, and 24.

FONTDLG Field - pfnDlgProc

pfnDlgProc (PFNDLGPROC)

Custom dialog procedure.

NULL unless the caller is subclassing the font dialog. When non-NULL, it points to the dialog procedure of the application.

FONTDLG Field - pszFamilyname

pszFamilyname (PSZ)

Family name buffer.

Buffer provided by the application for passing the family name of the font. The font family name used by the application to select a font. When the first character in this string is NULL, no family name was initially selected, and the dialog defaults to the system font.

A buffer must be passed to the font dialog to allow the dialog to return the selected font family name. The size of this buffer is placed in the *usFamilyBufLen* field.

FONTDLG Field - fxPointSize

fxPointSize (FIXED)

Point size of the font.

If FNTS_OWNERDRAWPREVIEW is set, 0 means the user wants to leave the font size unchanged and the application must update the preview area.

FONTDLG Field - fl

fl ([ULONG](#))

FNTS_* flags.

FNTS_APPLYBUTTON

An Apply push button is added to the dialog. This is useful in a modeless dialog.

FNTS_BITMAPONLY

The dialog presents bit-map fonts only. An application that changes fonts by using the presentation parameters (PP_* values) could use this flag.

FNTS_CENTER

The dialog is positioned in the center of its parent window, overriding any specified x,y position.

FNTS_CUSTOM

A custom dialog template is used to create the dialog. The *hMod* and *usDlgId* fields must be initialized.

FNTS_FIXEDWIDTHONLY

The dialog presents fixed-width (monospace) fonts only.

FNTS_HELPBUTTON

A Help push button of style (BS_HELP|BS_NOINTERFOCUS) with an ID of DID_HELP_BUTTON is added to the dialog. If the push button is pressed, a [WM_HELP](#) message is sent to the parameter of the [WinFontDlg](#) function call.

FNTS_INITFROMFATTRS

The dialog initializes itself from the font attribute structure ([FATTRS](#)) that is passed.

FNTS_MODELESS

The dialog is modeless; [WinFontDlg](#) returns immediately after creating the dialog window and returns the window handle to the application. The application should treat the dialog as if it were created with [WinLoadDlg](#). As in the modal (default) dialog case, the return value is found in the *lReturn* field of the FONTDLG structure passed to [WinFontDlg](#).

FNTS_NOSYNTHESIZEDFONTS

The dialog does not synthesize any fonts.

FNTS_OWNERDRAWPREVIEW

This flag makes the check boxes in the font dialog three-state check boxes, enabling the user to leave certain style attributes unchanged. Additionally, a [WM_DRAWITEM](#) message will be sent to the owner, providing the owner an opportunity to draw the preview window itself.

FNTS_PROPORTIONALONLY

The dialog presents proportionally spaced fonts only.

FNTS_RESETBUTTON

A Reset push button is added to the dialog. When this push button is pressed, the values for the dialog are restored to their initial values.

FNTS_VECTORONLY

The dialog presents vector fonts only.

FONTDLG Field - flFlags

flFlags ([ULONG](#))

FNTF_* flags.

FNTF_NOVIEWPRINTERFONTS

This flag is initialized only when both *hpsScreen* and *hpsPrinter* are not NULLHANDLE. On input, this parameter determines whether the printer fonts are to be included in the font list box. The user controls this with a check box.

FNTF_NOVIEWSCREENFONTS

This flag is initialized only when both *hpsScreen* and *hpsPrinter* are not NULLHANDLE. On input, this parameter determines whether the screen fonts should be included in the font list box. The user controls this with a check box.

FNTF_PRINTERFONTSELECTED

This determines if a printer-specific font is selected by the user. The application should make an approximation of this printer font when outputting to the screen. This is an output-only flag and is ignored on input.

FNTF_SCREENFONTSELECTED

This determines if a screen-specific font is selected by the user. The application should make an approximation of this screen font when outputting to the screen. This is an output-only flag and is ignored on input.

FONTDLG Field - flType

flType (ULONG)
The selected type bits.

These flags specify what additional attributes the user specified for the font. This field is used as the *flOptions* field in the **FACENAMEDESC** structure for GpiQueryFaceString.

FONTDLG Field - flTypeMask

flTypeMask (ULONG)
Mask of type bits to use.

This field is used only if FNTS_OWNERDRAWPREVIEW is specified. It tells which flags of the *flTypeMask* field the user wants to change, and is relevant only if the text for which the font is selected has different faces and styles.

FONTDLG Field - flStyle

flStyle (ULONG)
Selected style bits.

Flags for any additional selections the user specified for the font. This field is used as the *fsSelection* field in the **FATTRS** structure passed to GpiCreateLogFont.

FONTDLG Field - flStyleMask

flStyleMask (ULONG)
Mask of style bits to use.

This field is used only if FNTS_OWNERDRAWPREVIEW is specified. It tells which flags of the *flStyle* field the user wants to change and is relevant only if the text for which the font is selected has different faces and styles.

FONTDLG Field - clrFore

clrFore (LONG)
Font foreground color.

Foreground color of the font. This color is a value used for the color mode that *hpsScreen* is in. If FNTS_OWNERDRAWPREVIEW is specified, this value can be CLR_NOINDEX, leaving the foreground color "as is".

FONTDLG Field - clrBack

clrBack ([LONG](#))

Font background color.

Background color of the font. This color is a value used for the color mode that *AppScreen* is in. If FNTS_OwnerDrawPreview is specified, this value can be CLR_NOINDEX leaving the background color "as is".

FONTDLG Field - ulUser

ulUser ([ULONG](#))

Application-defined.

A [ULONG](#) that an application uses to store its state information when it is subclassing the font dialog.

FONTDLG Field - IReturn

IReturn ([LONG](#))

Return value.

Return value from [WinFontDlg](#). This value is the ID of the push button pressed to dismiss the dialog, DID_OK or DID_CANCEL, unless the application supplied additional push buttons in its template.

FONTDLG Field - ISRC

ISRC ([LONG](#))

System return code.

This field contains an FNTS_ERR return code. When a dialog fails, this field is used to tell the application the reason for the failure.

FONTDLG Field - IEmHeight

IEmHeight ([LONG](#))

Em height.

The Em height of the current font. This is the same as in the [FONTMETRICS](#) structure. It is an output-only parameter and its value has no effect on the behavior of the font dialog, but is updated when the user dismisses the dialog.

FONTDLG Field - IXHeight

IXHeight ([LONG](#))
X height.

The x height of the current font. This is the same as in the [FONTMETRICS](#) structure. It is an output-only parameter and its value has no effect on the behavior of the font dialog, but is updated when the user dismisses the dialog.

FONTDLG Field - IExternalLeading

IExternalLeading ([LONG](#))
External leading.

The external leading of the font. This is the same as in the [FONTMETRICS](#) structure. It is an output-only parameter and its value has no effect on the behavior of the font dialog, but is updated when the user dismisses the dialog.

FONTDLG Field - hMod

hMod ([HMODULE](#))
Module for custom dialog resources.

If FNTS_CUSTOM is set, this is the HMODULE from which the custom font dialog template is loaded. NULLHANDLE causes the dialog resource to be pulled from the module of the current EXE.

FONTDLG Field - fAttrs

fAttrs ([FATTRS](#))
Font-attribute structure.

Font-attribute structure of selected font. The [FATTRS](#) for the selected font. This is output-only for all fields except *usCodePage*, which is input/output, and the initial code page value passed is used for font selection. The value returned is the one for the matching font.

FONTDLG Field - sNominalPointSize

sNominalPointSize ([SHORT](#))
Font point size.

The nominal point size of the font. This is the same as in the [FONTMETRICS](#) structure. It is an output-only parameter and its value has no effect on the behavior of the font dialog, but is updated when the user dismisses the dialog.

FONTDLG Field - usWeight

usWeight ([USHORT](#))
Font weight.

The weight of the font. This is the weight-class/boldness the user selects for the font. This field is used as the *usWeightClass* field in the [FACENAMEDESC](#) structure for GpiQueryFaceString. When FNTS_OWNERDRAWPREVIEW is set, 0 causes the application to leave the font weight "as is" and the application must update the preview area.

FONTDLG Field - usWidth

usWidth ([USHORT](#))
Font width.

The width of the font. This is the width-class the user selects for the font. This field is used as the *usWidthClass* field in the [FACENAMEDESC](#) structure for GpiQueryFaceString. When FNTS_OWNERDRAWPREVIEW is set, 0 causes the application to leave the font width "as is" and the application must update the preview area.

FONTDLG Field - x

x ([SHORT](#))
The x-axis dialog position.

This, along with *y* and is used to position the dialog. It is updated in the structure if the user moves the dialog to a new position. This way, the dialog appears in the position at which it was left each time it is invoked. The FNTS_CENTER flag overrides this position and automatically centers the dialog in its parent.

FONTDLG Field - y

y ([SHORT](#))
The y-axis dialog position.

This, along with *x* and is used to position the dialog. It is updated in the structure if the user moves the dialog to a new position. This way, the dialog appears in the position at which it was left each time it is invoked. The FNTS_CENTER flag overrides this position and automatically centers the dialog in its parent.

FONTDLG Field - usDlgId

usDlgId (USHORT)
Dialog ID.

This sets the ID of the dialog window. If FNTS_CUSTOM is set, this is the ID of the resource that contains the custom dialog template.

FONTDLG Field - usFamilyBufLen

usFamilyBufLen (USHORT)
Buffersize.

Size of the buffer passed in the *pszFamilyname* resource that contains the custom dialog template.

FONTDLG Field - usReserved

usReserved (USHORT)
Reserved.

This is a reserved field.

FONTMETRICS

Font-metrics structure.

This structure is returned to applications on the GpiQueryFonts and GpiQueryFontMetrics calls and conveys information from the font creator to the application.

```
typedef struct _FONTMETRICS {
    CHAR        szFamilyname[FACE_SIZE]; /* Family name. */
    CHAR        szFacename[FACE_SIZE];   /* Face name. */
    USHORT      idRegistry;               /* Registry identifier. */
    USHORT      usCodePage;               /* Code page. */
    LONG        lEmHeight;                /* Em height. */
    LONG        lXHeight;                 /* X height. */
    LONG        lMaxAscender;             /* Maximum ascender. */
    LONG        lMaxDescender;            /* Maximum descender. */
    LONG        lLowercaseAscent;         /* Lowercase ascent. */
    LONG        lLowercaseDescent;        /* Lowercase descent. */
    LONG        lInternalLeading;           /* Internal leading. */
    LONG        lExternalLeading;          /* External leading. */
    LONG        lAveCharWidth;            /* Average character width. */
    LONG        lMaxCharInc;              /* Maximum character increment. */
    LONG        lEmInc;                   /* Em increment. */
    LONG        lMaxBaselineExt;          /* Maximum baseline extent. */
    SHORT       sCharSlope;               /* Character slope. */
    SHORT       sInlineDir;               /* Inline direction. */
    SHORT       sCharRot;                 /* Character rotation. */
    USHORT      usWeightClass;            /* Weight class. */
    USHORT      usWidthClass;             /* Width class. */
    SHORT       sXDeviceRes;              /* X-device resolution. */
    SHORT       sYDeviceRes;              /* Y-device resolution. */
}
```

```

SHORT    sFirstChar;          /* First character. */
SHORT    sLastChar;          /* Last character. */
SHORT    sDefaultChar;       /* Default character. */
SHORT    sBreakChar;         /* Break character. */
SHORT    sNominalPointSize;  /* Nominal point size. */
SHORT    sMinimumPointSize;  /* Minimum point size. */
SHORT    sMaximumPointSize;  /* Maximum point size. */
USHORT   fsType;             /* Type indicators. */
USHORT   fsDefn;             /* Definition indicators. */
USHORT   fsSelection;        /* Selection indicators. */
USHORT   fsCapabilities;     /* Font capabilities. */
LONG     lSubscriptXSize;    /* Subscript x-size. */
LONG     lSubscriptYSize;    /* Subscript y-size. */
LONG     lSubscriptXOffset;  /* Subscript x-offset. */
LONG     lSubscriptYOffset;  /* Subscript y-offset. */
LONG     lSuperscriptXSize;  /* Superscript x-size. */
LONG     lSuperscriptYSize;  /* Superscript y-size. */
LONG     lSuperscriptXOffset; /* Superscript x-offset. */
LONG     lSuperscriptYOffset; /* Superscript y-offset. */
LONG     lUnderscoreSize;    /* Underscore size. */
LONG     lUnderscorePosition; /* Underscore position. */
LONG     lStrikeoutSize;     /* Strikeout size. */
LONG     lStrikeoutPosition; /* Strikeout position. */
SHORT    sKerningPairs;      /* Kerning pairs. */
SHORT    sFamilyClass;       /* Font family design classification. */
LONG     lMatch;             /* Matched font identity. */
LONG     FamilyNameAtom;     /* Font family name atom. */
LONG     FaceNameAtom;       /* Font facename atom. */
PANOSE   panose;             /* Panose font descriptor. */
} FONTMETRICS;

typedef FONTMETRICS *PFONTMETRICS;

```

FONTMETRICS Field - szFamilyname[FACESIZE]

szFamilyname[FACESIZE] (CHAR)
Family name.

The family name of the font that describes the basic appearance of the font, for example, Times New Roman** This string is null terminated, and therefore is limited to 31 characters in length. Longer names may be retrieved by using the *FamilyNameAtom* field to retrieve the full name from the System Atom table.

FONTMETRICS Field - szFacename[FACESIZE]

szFacename[FACESIZE] (CHAR)
Face name.

The typeface name that defines the particular font, for example, Times New Roman Bold Italic. This string is null terminated, and therefore is limited to 31 characters in length. Longer names may be retrieved by using the *FaceNameAtom* field to retrieve the full name from the System Atom table.

FONTMETRICS Field - idRegistry

idRegistry ([USHORT](#))
Registry identifier.

The IBM registered number (or zero).

FONTMETRICS Field - usCodePage

usCodePage ([USHORT](#))
Code page.

Defines the registered code page supported by the font. For example, the original IBM PC code page is 437. A value of 0 implies that the font may be used with any of the OS/2 supported code pages.

Where a font contains special symbols for which there is no registered code page, then code page 65400 is used.

FONTMETRICS Field - IEmHeight

IEmHeight ([LONG](#))
Em height.

The height of the Em square in world coordinate units. This corresponds to the point size for the font.

FONTMETRICS Field - IXHeight

IXHeight ([LONG](#))
X height.

The nominal height above the baseline for lowercase characters (ignoring ascenders) in world coordinate units.

FONTMETRICS Field - IMaxAscender

IMaxAscender ([LONG](#))
Maximum ascender.

The maximum height above the baseline reached by any part of any symbol in the font in world coordinate units. This field may exceed *IEmHeight*.

FONTMETRICS Field - IMaxDescender

IMaxDescender (LONG)
Maximum descender.

The maximum depth below the baseline reached by any part of any symbol in the font in world coordinate units. This field may exceed *EmHeight*.

FONTMETRICS Field - ILowerCaseAscent

ILowerCaseAscent (LONG)
Lowercase ascent.

The maximum height above the baseline reached by any part of any lowercase (Latin unaccented "a" through "z") symbol in the font in world coordinate units.

FONTMETRICS Field - ILowerCaseDescent

ILowerCaseDescent (LONG)
Lowercase descent.

The maximum depth below the baseline reached by any part of any lowercase (Latin unaccented "a" through "z") symbol in the font in world coordinate units.

FONTMETRICS Field - IInternalLeading

IInternalLeading (LONG)
Internal leading.

The amount of space which, when subtracted from *IMaxAscender*, gives a font-design dependent, but glyph-set independent, measure of the distance above the baseline that characters extend. This calculation approximates the visual top to a row of characters without actually looking at the characters in the row.

For optimum results, this field should be used by applications to position the first line of a block of text by subtracting it from *IMaxAscender* and positioning the baseline that distance below whatever is above the text.

Note: This does not guarantee that characters will not overwrite information above them, but does give a font designer's view of where to place the text. Collision should be tested for, and additional space allocated if necessary.

FONTMETRICS Field - IExternalLeading

IExternalLeading (LONG)
External leading.

The amount of guaranteed white space advised by the font designer to appear between adjacent rows of text. This value may be zero.

Note: The fonts built in to Presentation Manager have zero in this field.

FONTMETRICS Field - IAveCharWidth

IAveCharWidth (LONG)
Average character width.

This is determined by multiplying the width of each lowercase character by a constant, adding the products, and then dividing by 1000. The letters involved in this, plus their constants, are as follows:

Letter	Constant
a	64
b	14
c	27
d	35
e	100
f	20
g	14
h	42
i	63
j	3
k	6
l	35
m	20
n	56
o	56
p	17
q	4
r	49
s	56
t	71
u	31
v	10
w	18
x	3
y	18
z	2
space	166

Note: For fixed pitch fonts, this value will be the same as the (A width + B width + C width) escapement of each character.

FONTMETRICS Field - IMaxCharInc

IMaxCharInc (LONG)
Maximum character increment.

The maximum character increment for the font in world coordinate units.

FONTMETRICS Field - IEmInc

IEmlnc (LONG)
Em increment.

The width of the Em square in world coordinate units. This corresponds to the point size of the font. When the horizontal device resolution equals the vertical device resolution this is equal to the em height.

FONTMETRICS Field - IMaxBaselineExt

IMaxBaselineExt (LONG)
Maximum baseline extent.

The maximum vertical space occupied by the font, in world coordinate units. This is the sum of *IMaxAscender* and *IMaxDescender* if both are positive. It is also the sum of *IInternalLeading* and *IEmHeight*.

One possible type of line spacing can be computed by adding *IMaxBaselineExt* to *IExternalLeading*. Such a line spacing, however, would be dependent on the glyph set included in the font. If a new version of the font should be made available, with new glyphs, then it is possible that this value will change because one of the new glyphs has gone above the previous *IMaxAscender* or below the previous *IMaxDescender*. More sophisticated applications will base line spacing on the point size (*IEmHeight*) of the font, which is an invariant of the font, multiplied by some factor (for example, 120%) plus any external leading.

This field may exceed *IEmHeight*.

FONTMETRICS Field - sCharSlope

sCharSlope (SHORT)
Character slope.

Defines the nominal slope for the characters of a font. The slope is defined in degrees increasing clockwise from the vertical. An *italic* font is an example of a font with a nonzero slope.

Note: The units for this metric are degrees and minutes, encoded as shown in the following example:

```
180 degrees 59 minutes would be represented as :  
  
    < byte 1  >      < byte 2  >  
  
    < Minutes >      < Degrees >  
  
0 1 1 1 0 1 1   0 1 0 1 1 0 1 0 0  
  
    59 min          180 degrees
```

FONTMETRICS Field - sInlineDir

sInlineDir (SHORT)

Inline direction.

The direction in which the characters in the font are designed for viewing. The direction is defined in degrees increasing clockwise from the horizontal (left-to-right). Characters are added to a line of text in the inline direction.

Note: The units for this metric are degrees and minutes, encoded as shown in *sCharSlope*.

FONTMETRICS Field - sCharRot

sCharRot (SHORT)

Character rotation.

The rotation of the character glyphs with respect to the baseline, the angle increasing counter clockwise. This is the angle assigned by the font designer.

Note: The units for this metric are degrees and minutes, encoded as shown in *sCharSlope*.

FONTMETRICS Field - usWeightClass

usWeightClass (USHORT)

Weight class.

Indicates the visual weight (thickness of strokes) of the characters in the font:

	Value	Description
1		Ultra-light
2		Extra-light
3		Light
4		Semi-light
5		Medium (normal)
6		Semi-bold
7		Bold
8		Extra-bold
9		Ultra-bold

FONTMETRICS Field - usWidthClass

usWidthClass (USHORT)

Width class.

Indicates the relative aspect ratio of the characters of the font in relation to the normal aspect ratio for this type of font:

Value	Description	% of normal width
1	Ultra-condensed	50
2	Extra-condensed	62.5
3	Condensed	75
4	Semi-condensed	87.5

5	Medium (normal)	100
6	Semi-expanded	112.5
7	Expanded	125
8	Extra-expanded	150
9	Ultra-expanded	200

FONTMETRICS Field - sXDeviceRes

sXDeviceRes ([SHORT](#))
X-device resolution.

For bit-map fonts this is the resolution in the X direction of the intended target device, measured in pels per inch.

For outline fonts this is the number of notional units in the X direction of the Em square, measured in notional units per Em. (Notional units are the units in which the outline is defined.)

FONTMETRICS Field - sYDeviceRes

sYDeviceRes ([SHORT](#))
Y-device resolution.

For bit-map fonts this is the resolution in the Y direction of the intended target device, measured in pels per inch.

For outline fonts this is the number of notional units in the Y direction of the Em square, measured in notional units per Em. (Notional units are the units in which the outline is defined.)

FONTMETRICS Field - sFirstChar

sFirstChar ([SHORT](#))
First character.

The code point of the first character in the font.

FONTMETRICS Field - sLastChar

sLastChar ([SHORT](#))
Last character.

The code point of the last character in the font, expressed as an offset from *sFirstChar*.

All code points between the first and last character specified must be supported by the font.

FONTMETRICS Field - sDefaultChar

sDefaultChar (SHORT)

Default character.

The code point that is used if a code point outside the range supported by the font is used, expressed as an offset from *sFirstChar*.

FONTMETRICS Field - sBreakChar

sBreakChar (SHORT)

Break character.

The code point that represents the "space" or "break" character for this font, expressed as an offset from *sFirstChar*. For example, if the first character is the space in code page 850, *sFirstChar* = 32, and *sBreakChar* = 0.

FONTMETRICS Field - sNominalPointSize

sNominalPointSize (SHORT)

Nominal point size.

For a bit-map font, this field contains the height of the font.

For an outline font, this field contains the height intended by the font designer. For example, some fonts are designed for text use in which case a value of 120 (12 point) would probably be placed in this field, whereas other fonts are designed for "display" use ("display" is typographer's terminology for larger sizes). This is not the only size at which the font can be used.

Measured in decipoints (a decipoint is 1/720th of an inch).

FONTMETRICS Field - sMinimumPointSize

sMinimumPointSize (SHORT)

Minimum point size.

For a bit-map font, this field does not apply. For an outline font, this field contains the minimum height intended by the font designer. Note that this is not a restriction of the size at which the font can be used.

Measured in decipoints (a decipoint is 1/720th of an inch).

FONTMETRICS Field - sMaximumPointSize

sMaximumPointSize (SHORT)
Maximum point size.

For a bit-map font, this field does not apply.

For an outline font, this field contains the maximum height intended by the font designer. Note that this is not a restriction of the size at which the font can be used.

Measured in decipoints (a decipoint is 1/720th of an inch).

FONTMETRICS Field - fsType

fsType (USHORT)
Type indicators.

This field contains the following information:

FM_TYPE_FIXED	Characters in the font have the same fixed width.
FM_TYPE_LICENSED	Licensed (protected) font.
FM_TYPE_KERNING	Font contains kerning information.
FM_TYPE_64K	Font is larger than 64KB (KB equals 1024 bytes) in size. If the following two bits are false, the font is for single-byte code pages. One of the bits may be set.
FM_TYPE_DBCS	Font is for double-byte code pages.
FM_TYPE_MBCS	Font is for mixed single- or double-byte code pages.
FM_TYPE_FACETRUNC	Font <i>szFacename</i> has been truncated.
FM_TYPE_FAMTRUNC	Font <i>szFamilyname</i> has been truncated.
FM_TYPE_ATOMS	The System Atom table atom values in <i>FamilyNameAtom</i> and in <i>FaceNameAtom</i> are valid.

FONTMETRICS Field - fsDefn

fsDefn (USHORT)
Definition indicators.

Contains the following font definition data:

FM_DEFN_OUTLINE	Font is a vector (outline) font; otherwise, it is a bit-map font.
FM_DEFN_GENERIC	Font is in a format that can be used by the GPI; otherwise, it is a device font.

FONTMETRICS Field - fsSelection

fsSelection (USHORT)
Selection indicators.

Contains information about the font patterns in the physical font.

Note: The flags do not reflect simulations applied to the physical font.

Possible values are:

FM_SEL_ITALIC	True indicates that this font is designed as an italic font.
FM_SEL_UNDERSCORE	TRUE indicates that this font is designed with underscores included in each character.
FM_SEL_NEGATIVE	TRUE indicates that this font is designed with the background and foreground reversed.
FM_SEL_OUTLINE	TRUE indicates that this font is designed with outline (hollow) characters.
FM_SEL_STRIKEOUT	TRUE indicates that this font is designed with an overstrike through each character.
FM_SEL_BOLD	TRUE indicates that this font is designed with bold characters.
FM_SEL_ISO9241_TESTED	This flag indicates that the font has been tested for compliance to ISO 9241. The presence of this flag doesn't indicate whether the font passed or failed, only that it was tested. Note: While the fonts were primarily tested for meeting the ISO standard, they have also been designed to meet the German standard DIN 66 234. Where the two standards differ, the fonts have been designed to meet the more stringent requirement.

FONTMETRICS Field - fsCapabilities

fsCapabilities (USHORT)
Font capabilities.

This attribute applies only to device fonts.

FM_CAP_NOMIX	Characters may not be mixed with graphics.										
QUALITY	The most significant byte may contain the following numeric value: <table><tr><td>0</td><td>Undefined</td></tr><tr><td>1</td><td>DP quality</td></tr><tr><td>2</td><td>DP draft</td></tr><tr><td>3</td><td>Near Letter Quality</td></tr><tr><td>4</td><td>Letter Quality</td></tr></table>	0	Undefined	1	DP quality	2	DP draft	3	Near Letter Quality	4	Letter Quality
0	Undefined										
1	DP quality										
2	DP draft										
3	Near Letter Quality										
4	Letter Quality										

FONTMETRICS Field - ISubscriptXSize

ISubscriptXSize (LONG)
Subscript x-size.

The horizontal size recommended by the font designer for subscripts for this font in world coordinate units.

FONTMETRICS Field - ISubscriptYSize

ISubscriptYSize (LONG)
Subscript y-size.

The vertical size recommended by the font designer for subscripts for this font in world coordinate units.

FONTMETRICS Field - ISubscriptXOffset

ISubscriptXOffset (LONG)
Subscript x-offset.

The baseline x-offset recommended by the font designer for subscripts for this font in world coordinate units.

FONTMETRICS Field - ISubscriptYOffset

ISubscriptYOffset (LONG)
Subscript y-offset.

The baseline y-offset recommended by the font designer for subscripts for this font in world coordinate units.

Note: Positive numbers indicate an offset below the baseline.

FONTMETRICS Field - ISuperscriptXSize

ISuperscriptXSize (LONG)
Superscript x-size.

The horizontal size recommended by the font designer for superscripts for this font in world coordinate units.

FONTMETRICS Field - ISuperscriptYSize

ISuperscriptYSize (LONG)
Superscript y-size.

The vertical point size recommended by the font designer for superscripts for this font in world coordinate units.

FONTMETRICS Field - ISuperscriptXOffset

ISuperscriptXOffset (LONG)
Superscript x-offset.

The baseline x-offset recommended by the font designer for superscripts for this font in world coordinate units.

FONTMETRICS Field - ISuperscriptYOffset

ISuperscriptYOffset (LONG)
Superscript y-offset.

The baseline y-offset recommended by the font designer for superscripts for this font in world coordinate units.

FONTMETRICS Field - IUnderscoreSize

IUnderscoreSize (LONG)
Underscore size.

The width (thickness) of the underscore stroke in world coordinate units. This describes the actual underscore in the font if FM_SEL_UNDERSCORE is also set. Otherwise it describes what the engine will simulate if underscore is requested in GpiCreateLogFont.

FONTMETRICS Field - IUnderscorePosition

IUnderscorePosition (LONG)
Underscore position.

The position of the underscore stroke from the baseline in world coordinate units. This describes the actual underscore in the font if FM_SEL_UNDERSCORE is also set. Otherwise it describes what the engine will simulate if underscore is requested in GpiCreateLogFont.

Note: Positive values indicate an offset below the baseline.

FONTMETRICS Field - IStrikeoutSize

IStrikeoutSize (LONG)

Strikeout size.

The width of the strikeout stroke in world coordinate units. This describes the actual underscore in the font if FM_SEL_STRIKEOUT is also set. Otherwise it describes what the engine will simulate if overstrike is requested in GpiCreateLogFont.

FONTMETRICS Field - IStrikeoutPosition

IStrikeoutPosition (LONG)

Strikeout position.

The position of the strikeout stroke relative to the baseline in world coordinate units. This describes the actual underscore in the font if FM_SEL_STRIKEOUT is also set. Otherwise it describes what the engine will simulate if overstrike is requested in GpiCreateLogFont.

FONTMETRICS Field - sKerningPairs

sKerningPairs (SHORT)

Kerning pairs.

The number of kerning pairs in the kerning pair table.

FONTMETRICS Field - sFamilyClass

sFamilyClass (SHORT)

Font family design classification.

This value contains a font class and its subclass.

FONTMETRICS Field - IMatch

IMatch (LONG)

Matched font identity.

This uniquely identifies the font for a given device and device driver combination. A positive match number signifies that the font is a generic (engine) font while a negative number indicates a device font (a native or downloadable font). This value should not be used

to identify a font across system boundaries.

FONTMETRICS Field - FamilyNameAtom

FamilyNameAtom (LONG)

Font family name atom.

This value contains the atom identifier for the font family name in the System Atom Table.

FONTMETRICS Field - FaceNameAtom

FaceNameAtom (LONG)

Font facename atom.

This value contains the atom identifier for the font face name in the System Atom Table.

FONTMETRICS Field - panose

panose (PANOSE)

Panose font descriptor.

This is the Panose descriptor identifying the visual characteristics of the font.

FRAMECDATA

Frame-control data structure.

```
typedef struct _FRAMECDATA {
    USHORT      cb; /* Length. */
    ULONG       flCreateFlags; /* Frame-creation flags. */
    USHORT      hmodResources; /* Identifier of required resource. */
    USHORT      idResources; /* Resource identifier. */
} FRAMECDATA;

typedef FRAMECDATA *PFRAMECDATA;
```

FRAMECDATA Field - cb

cb (**USHORT**)
Length.

FRAMECDATA Field - flCreateFlags

flCreateFlags (**ULONG**)
Frame-creation flags.

Possible values are described in the following list:

- FCF_TITLEBAR
- FCF_SYSMENU
- FCF_MENU
- FCF_SIZEBORDER
- FCF_MINBUTTON
- FCF_MAXBUTTON
- FCF_MINMAX
- FCF_VERTSCROLL
- FCF_HORZSCROLL
- FCF_DLGBORDER
- FCF_BORDER
- FCF_SHELLPOSITION
- FCF_TASKLIST
- FCF_NOBYTEALIGN
- FCF_NOMOVEWITHOWNER
- FCF_ICON
- FCF_ACCELTABLE
- FCF_SYSMODAL
- FCF_SCREENALIGN
- FCF_MOUSEALIGN
- FCF_HIDEBUTTON
- FCF_HIDEMAX
- FCF_AUTOICON
- FCF_DBE_APPSTAT
- FCF_STANDARD

The standard setting is equivalent to setting FCF_TITLEBAR, FCF_SYSMENU, FCF_MENU, FCF_SIZEBORDER, FCF_MINMAX, FCF_ICON, FCF_ACCELTABLE, FCF_SHELLPOSITION, and FCF_TASKLIST.

FRAMECDATA Field - hmodResources

hmodResources ([USHORT](#))

Identifier of required resource.

This is supplied in an environment-dependent manner.

FRAMECDATA Field - idResources

idResources ([USHORT](#))

Resource identifier.

GETPORTFROMQ

Information about the ports connected to the specified queue.

```
typedef struct _GETPORTFROMQ {  
    ULONG        ulPorts; /* Number of ports connected to this print queue. */  
    PORTFROMQ    PortFromQ[1]; /* An array (size ulPorts) of ports connected to this queue. */  
} GETPORTFROMQ;  
  
typedef GETPORTFROMQ *PGETPORTFROMQ;
```

GETPORTFROMQ Field - ulPorts

ulPorts ([ULONG](#))

Number of ports connected to this print queue.

GETPORTFROMQ Field - PortFromQ[1]

PortFromQ[1] ([PORTFROMQ](#))

An array (size *ulPorts*) of ports connected to this queue.

This array is a list of PORTFROMQ structures.

GRADIENTL

Direction-vector structure.

```
typedef struct _GRADIENL {
    LONG    x; /* X-component of direction. */
    LONG    y; /* Y-component of direction. */
} GRADIENL;

typedef GRADIENL *PGRADIENL;
```

GRADIENL Field - x

x (LONG)
X-component of direction.

GRADIENL Field - y

y (LONG)
Y-component of direction.

GRIDINFO

This structure is used by [CM_QUERYGRIDINFO](#) and [CM_SETGRIDINFO](#) messages to query and set grid characteristics. These characteristics are used with the grid arrange options or when the view is CV_GRID.

```
typedef struct _GRIDINFO {
    ULONG    cb; /* Length of the GRIDINFO structure in bytes. */
    ULONG    cxGrid; /* Grid square width. */
    ULONG    cyGrid; /* Grid square height. */
    SHORT    sGridRows; /* Number of rows in the grid. */
    SHORT    sGridCols; /* Number of columns in the grid. */
    ULONG    cGridSquares; /* Number of available grid squares. */
    PGRIDSQUARE pGrid; /* Pointer to array of GRIDSQUARE structures. */
} GRIDINFO;

typedef GRIDINFO *PGRIDINFO;
```

GRIDINFO Field - cb

cb (ULONG)
Length of the GRIDINFO structure in bytes.

GRIDINFO Field - cxGrid

cxGrid ([ULONG](#))
Grid square width.

GRIDINFO Field - cyGrid

cyGrid ([ULONG](#))
Grid square height.

GRIDINFO Field - sGridRows

sGridRows ([SHORT](#))
Number of rows in the grid.

GRIDINFO Field - sGridCols

sGridCols ([SHORT](#))
Number of columns in the grid.

GRIDINFO Field - cGridSquares

cGridSquares ([ULONG](#))
Number of available grid squares.

GRIDINFO Field - pGrid

pGrid ([PGRIDSQUARE](#))
Pointer to array of GRIDSQUARE structures.

If this field is non-NULL, the container expects the pointer to point to an array of GRIDQUARE structures, one for each available grid square in the grid. In this case fields *cxGrid* and *CyGrid* are ignored.

If this field is NULL, the container creates a grid with grid squares of width *cxGrid* and height *cyGrid*.

GRIDSQUARE

This structure is used to create an array that defines the arrangement of icons in a container. Each grid square in the array has a number that identifies its location in the container window, and a state that indicates whether it can be occupied by an icon.

The **CM_SETGRIDINFO** message is used to set the pattern.

```
typedef struct _GRIDSQUARE {
    ULONG    ulNumber;    /* Number of the grid square. */
    ULONG    ulState;     /* State of the grid square. */
    RECTL    rectlSquare; /* Coordinates of the grid square in the container window. */
} GRIDSQUARE;

typedef GRIDSQUARE *PGRIDSQUARE;
```

GRIDSQUARE Field - ulNumber

ulNumber (**ULONG**)

Number of the grid square.

Grid squares in the array are numbered, starting from 1. The order starts from the upper left corner of the window and goes from left-to-right, top-to-bottom.

GRIDSQUARE Field - ulState

ulState (**ULONG**)

State of the grid square.

This field contains one of the following values:

CMA_AVAIL

This grid square can hold an icon.

CMA_UNAVAIL

This grid square is not available.

GRIDSQUARE Field - rectlSquare

rectlSquare (**RECTL**)

Coordinates of the grid square in the container window.

HAB

Anchor-block handle.

```
typedef LHANDLE HAB;
```

HACCEL

Accelerator-table handle.

```
typedef LHANDLE HACCEL;
```

HAPP

Handle of an application.

```
typedef LHANDLE HAPP;
```

HATOMTBL

Atom-table handle.

```
typedef LHANDLE HATOMTBL;
```

HBITMAP

Bitmap handle.

```
typedef LHANDLE HBITMAP;
```

HCINFO

Hardcopy-capabilities structure.

```
typedef struct _HCINFO {
    CHAR    szFormname[32]; /* Form name. */
    LONG    cx;             /* Width (left-to-right) in millimeters. */
    LONG    cy;             /* Height (top-to-bottom) in millimeters. */
    LONG    xLeftClip;      /* Left clip limit in millimeters. */
    LONG    yBottomClip;    /* Bottom clip limit in millimeters. */
    LONG    xRightClip;     /* Right clip limit in millimeters. */
    LONG    yTopClip;       /* Top clip limit in millimeters. */
    LONG    xPels;          /* Number of pels between left and right clip limits. */
    LONG    yPels;          /* Number of pels between bottom and top clip limits. */
    LONG    flAttributes;   /* Attributes of the form identifier. */
} HCINFO;

typedef HCINFO *PHCINFO;
```

HCINFO Field - szFormname[32]

szFormname[32] (CHAR)
Form name.

HCINFO Field - cx

cx (LONG)
Width (left-to-right) in millimeters.

HCINFO Field - cy

cy (LONG)
Height (top-to-bottom) in millimeters.

HCINFO Field - xLeftClip

xLeftClip (LONG)
Left clip limit in millimeters.

HCINFO Field - yBottomClip

yBottomClip (LONG)
Bottom clip limit in millimeters.

HCINFO Field - xRightClip

xRightClip (LONG)
Right clip limit in millimeters.

HCINFO Field - yTopClip

yTopClip (LONG)
Top clip limit in millimeters.

HCINFO Field - xPels

xPels (LONG)
Number of pels between left and right clip limits.

HCINFO Field - yPels

yPels (LONG)
Number of pels between bottom and top clip limits.

HCINFO Field - flAttributes

flAttributes (LONG)
Attributes of the form identifier.

HCAPS_SELECTABLE

The form is installed on the printer as given by the printer properties dialog. It is available from an alternate form source without operator intervention. If the form does not have this bit set and is used (if the user selects it), a

HCAPS_CURRENT	"forms mismatch" error is generated by the printer object.
	The form is the one currently selected by the <i>pdriv</i> field (the job properties) in DEVOPENSTRUC

HDC

Device-context handle.

```
typedef LHANDLE HDC;
```

HDDF

Dynamic data formatting handle.

```
typedef VOID *HDDF;
```

HELPINIT

Help Manager initialization structure.

```
typedef struct _HELPINIT {
    ULONG      cb; /* Count of bytes of the initialization structure. */
    ULONG      ulReturnCode; /* Value returned by the Help Manager from initialization. */
    PSZ        pszTutorialName; /* Indicates to the Help Manager that the application has a tutorial. */
    HELPTABLE  htHelpTable; /* Help table. */
    HMODULE     hmodHelpTableModule; /* Resource file identity. */
    HMODULE     hmodAccelActionBarModule; /* Handle of the containing DLL. */
    ULONG      idAccelTable; /* Identity of the accelerator table. */
    ULONG      idActionBar; /* Identity of the action bar template used by the Help Manager. */
    PSZ        pszHelpWindowTitle; /* Window title for the main help window of this help instance. */
    ULONG      fShowPanelId; /* Show panel identity indicator. */
    PSZ        pszHelpLibraryName; /* Help panel library names. */
} HELPINIT;

typedef HELPINIT *PHELPINIT;
```

HELPINIT Field - cb

cb ([ULONG](#))
Count of bytes of the initialization structure.

HELPINIT Field - ulReturnCode

ulReturnCode (ULONG)
Value returned by the Help Manager from initialization.

0 Initialization was successful.

HELPINIT Field - pszTutorialName

pszTutorialName (PSZ)
Indicates to the Help Manager that the application has a tutorial program.

NULL The application either does not have a tutorial program, or the tutorial name is specified in each help panel definition.

Other Default tutorial name.

HELPINIT Field - htHelpTable

htHelpTable (HELPTABLE)
Help table.

The help table or the identity of the help table. If this is the identity of the help table in a resource file, the low-order word contains the identity of the table and the high-order word must be 0xFFFF.

The help table associates each application window with its help subtable and the identity of its extended help panel.

HELPINIT Field - hmodHelpTableModule

hmodHelpTableModule (HMODULE)
Resource file identity.

If the *htHelpTable* contains the identity of the help table, this field identifies the module handle returned by the DosLoadModule call by which the application loaded the resource file.

NULL The resource file containing the help table was appended to the application's .EXE file.

Other Resource file identity.

HELPINIT Field - hmodAccelActionBarModule

hmodAccelActionBarModule ([HMODULE](#))
Handle of the containing DLL.

The handle of the DLL which contains the accelerator table and action bar template to be used by the Help Manager.

NULL
Use the default action bar and accelerator table defined by the Help Manager.

Other
Handle of the DLL.

HELPINIT Field - idAccelTable

idAccelTable ([ULONG](#))
Identity of the accelerator table.

The accelerator table resides in the DLL provided in the *hmodAccelActionBarModule* field.

NULL
Use the default accelerator table.

Other
Identity of the accelerator table.

HELPINIT Field - idActionBar

idActionBar ([ULONG](#))
Identity of the action bar template used by the Help Manager.

The action bar template resides in the DLL provided in the *hmodAccelActionBarModule* field.

NULL
Use the default action bar.

Other
Identity of the action bar.

HELPINIT Field - pszHelpWindowTitle

pszHelpWindowTitle ([PSZ](#))
Window title for the main help window of this help instance.

HELPINIT Field - fShowPanelId

fShowPanelId (ULONG)

Show panel identity indicator.

The constants corresponding to the panel identity flags are in the PMHELP.H include file.

CMIC_SHOW_PANEL_ID

Show the panel identity on a help panel.

CMIC_HIDE_PANEL_ID

Do not show the panel identity on a help panel.

HELPINIT Field - pszHelpLibraryName

pszHelpLibraryName (PSZ)

Help panel library names.

The names of the help panel libraries that the Help Manager searches on each help request. The names must be separated by a blank.

The Help Manager looks for the libraries in the path set by the HELP environment variable. If the library is not found, the Help Manager will look for the libraries in the current directory.

HELPSUBTABLE

Help subtable.

A help subtable is an array of records, preceded by a value that specifies the size of each help-subtable record.

```
typedef USHORT _HELPSUBTABLE {
    USHORT    usSubitemSize;      /* Size of each record of the help subtable. */
    USHORT    HelpSubTableEntry[]; /* Help subtable records. */
};
```

The first entry in the help subtable indicates the size of the records that follow in the subtable. Each of the following entries in the help subtable is a record that consists of a Field ID parameter, a Help Panel ID parameter, and an optional array of application-related USHORT integers. The minimum number of words in the record is two: the Field ID and the Help Panel ID. The last record in the subtable must be a NULL entry.

The Field ID is the symbolic constant for a field from which the user can request help. The Field ID can identify a control, a menu item, or a message box, and must be unique across the help subtable. The value 0xFFFF is reserved for use by the Help Manager.

The Help Panel ID is the resource ID (res) of the contextual help panel to be associated with the field in the Field ID parameter. This is the panel to be displayed when the user requests help for the field.

The optional array of USHORT integers is ignored by the Help Manager and can be used to store information of relevance to the application.

There can be a maximum of 16,000 help subtables for a given help instance and each subtable can have a maximum of 64K bytes of data.

The following figure contains the declaration of a help subtable that contains only Field IDs and Help Panel IDs. In this subtable, each of the records after the size entry consists of 1 Field ID and 1 Help Panel ID for a size of 2. Note that the last record is filled with NULLs (0) to indicate the end of the array.

```
HELPSUBTABLE HelpSubTable[] =
{
    2,                                /* Size of each record */
    FIELD_ID_1, IDRES_HELP1,          /* The first record    */
    FIELD_ID_2, IDRES_HELP2,          /* The second record   */
    FIELD_ID_3, IDRES_HELP3,          /* The third record    */
    FIELD_ID_4, IDRES_HELP4,          /* The fourth record   */
    FIELD_ID_5, IDRES_HELP5,          /* The fifth record    */
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* The rest of the array is NULLs */
};
```

```

FIELD_ID_6, IDRES_HELP6, /* The sixth record */
0, 0 /* NULL record == end of the array */
}

```

HELPSUBTABLE Field - usSubitemSize

usSubitemSize (USHORT)

Size of each record of the help subtable.

This entry defines the number of parameters in each record in the rest of the help subtable. The minimum number of words in each record is two.

2

The minimum number of words in each record of the help subtable. This value is used when each help subtable record consists only of a Field ID and Help Panel ID.

Other

This value is used when a help subtable record consists of a Field ID, Help Panel ID, and an array of application-related USHORT integers.

HELPSUBTABLE Field - HelpSubTableEntry[]

HelpSubTableEntry[] (USHORT)

Help subtable records.

This is the array of help subtable records, each of which contains a Field ID, a Help Panel ID, and an optional array of application-related USHORT integers. The last record of the array must be a NULL entry.

HELPTABLE

Help table.

This is a collection of help table entries, each of which has the structure defined below, the last entry of the collection being a NULL structure.

```

typedef struct _HELPTABLE {
    USHORT          idAppWindow; /* Application window identity. */
    HELPSUBTABLE    phstHelpSubTable; /* Help subtable for this application window. */
    USHORT          idExtPanel; /* Identity of the extended help panel for the application window. */
} HELPTABLE;

typedef HELPTABLE *PHELPTABLE;

```

HELPTABLE Field - idAppWindow

idAppWindow ([USHORT](#))
Application window identity.

HELPTABLE Field - phstHelpSubTable

phstHelpSubTable ([PHELPSUBTABLE](#))
Help subtable for this application window.

HELPTABLE Field - idExtPanel

idExtPanel ([USHORT](#))
Identity of the extended help panel for the application window.

HENUM

Window-enumeration handle.
`typedef LHANDLE HENUM;`

HEV

Value (32-bit) used as an event semaphore handle.
`typedef ULONG HEV;`

HINI

Initialization-file handle.
`typedef LHANDLE HINI;`

HLIB

Library handle.

```
typedef HMODULE HLIB;
```

HMF

Metafile handle.

```
typedef LHANDLE HMF;
```

HMODULE

Module handle.

```
typedef LHANDLE HMODULE;
```

HMQ

Message-queue handle.

```
typedef LHANDLE HMQ;
```

HMTX

Value (32-bit) used as a mutex semaphore handle.

```
typedef ULONG HMTX;
```

HMUX

Value (32-bit) used as a muxwait semaphore handle.

```
typedef ULONG HMUX;
```

HOBJECT

Workplace object handle.

```
typedef LHANDLE HOBJECT;
```

HPOINTER

Pointer handle.

```
typedef LHANDLE HPOINTER;
```

HPROGRAM

Program handle.

```
typedef LHANDLE HPROGRAM;
```

HPS

Presentation-space handle.

```
typedef LHANDLE HPS;
```

HRGN

Region handle.

```
typedef LHANDLE HRGN;
```

HSAVEWP

Frame window-repositioning process handle.

```
typedef LHANDLE HSAVEWP;
```

HSEM

Semaphore handle.

```
typedef VOID *HSEM;
```

HSPL

Spooler handle.

```
typedef LHANDLE HSPL;
```

HSTR

String handle.

```
typedef LHANDLE HSTR;
```

HSWITCH

Switch-list entry handle.

```
typedef LHANDLE HSWITCH;
```

HWND

Window handle.

```
typedef LHANDLE HWND;
```

ICONINFO

Icon information data structure.

```
typedef struct _ICONINFO {
    ULONG      cb;           /* Length of the ICONINFO structure. */
    ULONG      fFormat;      /* Indicates where the icon resides. */
    PSZ        pszFileName; /* Name of the file containing icon data. */
    HMODULE    hmod;         /* Module containing the icon resource. */
    ULONG      resid;        /* Identity of the icon resource. */
    ULONG      cbIconData;   /* Length of the icon data in bytes. */
    PVOID      pIconData;   /* Pointer to the buffer containing icon data. */
} ICONINFO;

typedef ICONINFO *PICONINFO;
```

ICONINFO Field - cb

cb (**ULONG**)
Length of the ICONINFO structure.

ICONINFO Field - fFormat

fFormat (**ULONG**)
Indicates where the icon resides.

Possible values are:

ICON_FILE	Icon file supplied.
ICON_RESOURCE	Icon resource supplied.
ICON_DATA	Icon data supplied.
ICON_CLEAR	Go back to default icon.

ICONINFO Field - pszFileName

pszFileName (**PSZ**)
Name of the file containing icon data.

This value is ignored if *fFormat* is not equal to ICON_FILE.

ICONINFO Field - hmod

hmod ([HMODULE](#))

Module containing the icon resource.

This value is ignored if *ifFormat* is not equal to ICON_RESOURCE.

ICONINFO Field - resid

resid ([ULONG](#))

Identity of the icon resource.

This value is ignored if *ifFormat* is not equal to ICON_RESOURCE.

ICONINFO Field - cblIconData

cblIconData ([ULONG](#))

Length of the icon data in bytes.

This value is ignored if *ifFormat* is not equal to ICON_DATA.

ICONINFO Field - plconData

plconData ([PVOID](#))

Pointer to the buffer containing icon data.

This value is ignored if *ifFormat* is not equal to ICON_DATA.

IMAGEBUNDLE

Image-attributes bundle structure.

```
typedef struct _IMAGEBUNDLE {  
    LONG        lColor;           /* Image foreground color. */  
    LONG        lBackColor;       /* Image background color. */  
    USHORT      usMixMode;        /* Image foreground-mix mode. */  
    USHORT      usBackMixMode;    /* Image background-mix mode. */  
};
```

```
} IMAGEBUNDLE;  
typedef IMAGEBUNDLE *PIMAGEBUNDLE;
```

IMAGEBUNDLE Field - IColor

IColor ([LONG](#))
Image foreground color.

IMAGEBUNDLE Field - IBackColor

IBackColor ([LONG](#))
Image background color.

IMAGEBUNDLE Field - usMixMode

usMixMode ([USHORT](#))
Image foreground-mix mode.

IMAGEBUNDLE Field - usBackMixMode

usBackMixMode ([USHORT](#))
Image background-mix mode.

IPT

Insertion point for multi-line entry field.

```
typedef LONG IPT;
```

LBOXINFO

List box information structure.

```
typedef struct _LBOXINFO {  
    LONG    lItemIndex; /* Index of the item to insert after. */  
    ULONG    ulItemCount; /* Number of items to be inserted into the list. */  
    ULONG    reserved; /* Reserved value, must be 0. */  
    ULONG    reserved2; /* Reserved value, must be 0. */  
} LBOXINFO;  
  
typedef LBOXINFO *PLBOXINFO;
```

LBOXINFO Field - lItemIndex

lItemIndex (LONG)

Index of the item to insert after.

Possible values are described in the following list:

LIT_END

Add items to the end of the list.

LIT_SORTASCENDING

Add items to the list and sort the complete list in ascending order.

LIT_SORTDESCENDING

Add items to the list and sort the complete list in descending order.

Other

Add the items to the list after the specified zero-based index. Valid range is 0 to 32 767.

LBOXINFO Field - ulItemCount

ulItemCount (ULONG)

Number of items to be inserted into the list.

A maximum of 32 768 can be inserted into the list at one time.

LBOXINFO Field - reserved

reserved (ULONG)

Reserved value, must be 0.

LBOXINFO Field - reserved2

reserved2 (ULONG)
Reserved value, must be 0.

LHANDLE

The handle of a resource.

```
typedef unsigned long LHANDLE;
```

LINEBUNDLE

Line-attributes bundle structure.

```
typedef struct _LINEBUNDLE {  
    LONG      lColor;           /* Line foreground color. */  
    LONG      lBackColor;       /* Line background color. */  
    USHORT    usMixMode;        /* Line foreground-mix mode. */  
    USHORT    usBackMixMode;    /* Line background-mix mode. */  
    FIXED     fxWidth;          /* Line width. */  
    LONG      lGeomWidth;       /* Geometric line width. */  
    USHORT    usType;           /* Line type. */  
    USHORT    usEnd;            /* Line end. */  
    USHORT    usJoin;           /* Line join. */  
    USHORT    usReserved;       /* Reserved. */  
} LINEBUNDLE;  
  
typedef LINEBUNDLE *PLINEBUNDLE;
```

LINEBUNDLE Field - IColor

IColor (LONG)
Line foreground color.

LINEBUNDLE Field - IBackColor

IBackColor (LONG)
Line background color.

LINEBUNDLE Field - usMixMode

usMixMode (USHORT)
Line foreground-mix mode.

LINEBUNDLE Field - usBackMixMode

usBackMixMode (USHORT)
Line background-mix mode.

LINEBUNDLE Field - fxWidth

fxWidth (FIXED)
Line width.

LINEBUNDLE Field - lGeomWidth

lGeomWidth (LONG)
Geometric line width.

LINEBUNDLE Field - usType

usType (USHORT)
Line type.

LINEBUNDLE Field - usEnd

usEnd (USHORT)
Line end.

LINEBUNDLE Field - usJoin

usJoin ([USHORT](#))
Line join.

LINEBUNDLE Field - usReserved

usReserved ([USHORT](#))
Reserved.

LONG

Signed integer in the range -2 147 483 648 through 2 147 483 647.

```
#define LONG long
```

Note: Where this data type represents a graphic coordinate in world or model space, its value is restricted to -134 217 728 through 134 217 727.

A graphic coordinate in device or screen coordinates is restricted to -32 768 through 32 767.

The value of a graphic coordinate may be further restricted by any transforms currently in force, including the positioning of the origin of the window on the screen. In particular, coordinates in world or model space must not generate coordinate values after transformation (that is, in device or screen space) outside the range -32 768 through 32 767.

MARKERBUNDLE

Marker-attributes bundle structure.

```
typedef struct _MARKERBUNDLE {  
    LONG      lColor;           /* Marker foreground color. */  
    LONG      lBackColor;      /* Marker background color. */  
    USHORT    usMixMode;        /* Marker foreground-mix mode. */  
    USHORT    usBackMixMode;    /* Marker background-mix mode. */  
    USHORT    usSet;            /* Marker set. */  
    USHORT    usSymbol;         /* Marker symbol. */  
    SIZEF     sizfxCell;        /* Marker cell. */  
} MARKERBUNDLE;
```

```
typedef MARKERBUNDLE *PMARKERBUNDLE;
```

MARKERBUNDLE Field - lColor

IColor (LONG)
Marker foreground color.

MARKERBUNDLE Field - IBackColor

IBackColor (LONG)
Marker background color.

MARKERBUNDLE Field - usMixMode

usMixMode (USHORT)
Marker foreground-mix mode.

MARKERBUNDLE Field - usBackMixMode

usBackMixMode (USHORT)
Marker background-mix mode.

MARKERBUNDLE Field - usSet

usSet (USHORT)
Marker set.

MARKERBUNDLE Field - usSymbol

usSymbol (USHORT)
Marker symbol.

MARKERBUNDLE Field - sizfxCell

sizeCell ([SIZEF](#))
Marker cell.

MATRIXLF

Matrix-elements structure.

```
typedef struct _MATRIXLF {  
    FIXED    fxM11; /* First element of first row. */  
    FIXED    fxM12; /* Second element of first row. */  
    LONG     lM13;  /* Third element of first row. */  
    FIXED    fxM21; /* First element of second row. */  
    FIXED    fxM22; /* Second element of second row. */  
    LONG     lM23;  /* Third element of second row. */  
    LONG     lM31;  /* First element of third row. */  
    LONG     lM32;  /* Second element of third row. */  
    LONG     lM33;  /* Third element of third row. */  
} MATRIXLF;  
  
typedef MATRIXLF *PMATRIXLF;
```

MATRIXLF Field - fxM11

fxM11 ([FIXED](#))
First element of first row.

MATRIXLF Field - fxM12

fxM12 ([FIXED](#))
Second element of first row.

MATRIXLF Field - lM13

lM13 ([LONG](#))
Third element of first row.

MATRIXLF Field - fxM21

fxM21 (FIXED)
First element of second row.

MATRIXLF Field - fxM22

fxM22 (FIXED)
Second element of second row.

MATRIXLF Field - IM23

IM23 (LONG)
Third element of second row.

MATRIXLF Field - IM31

IM31 (LONG)
First element of third row.

MATRIXLF Field - IM32

IM32 (LONG)
Second element of third row.

MATRIXLF Field - IM33

IM33 (LONG)
Third element of third row.

MB2D

Array of button definitions.

```
typedef struct _MB2D {  
    CHAR      achText[MAX_MB2DTEXT+1]; /* Text of the button. */  
    ULONG     idButtons;                /* Button Id returned when selected. */  
    ULONG     flStyle;                  /* Button style flags. */  
} MB2D;  
  
typedef MB2D *PMB2D;
```

MB2D Field - achText[MAX_MB2DTEXT+1]

achText[MAX_MB2DTEXT+1] (CHAR)
Text of the button.

For example, "Cancel".

Currently, MAX_MB2DTEXT is equal to 70.

MB2D Field - idButtons

idButtons (ULONG)
Button Id returned when selected.

MB2D Field - flStyle

flStyle (ULONG)
Button style flags.

These style flags may be ORed with internal styles.

MB2INFO

Button information block.

```
typedef struct _MB2INFO {  
    ULONG     cb;                      /* Current size of the structure. */  
    HPOINTER  hIcon;                   /* Icon handle. */  
    ULONG     cButtons;                /* Number of buttons. */  
    ULONG     flStyle;                 /* Icon style flags. */  
    HWND      hwndNotify;              /* Owner notification handle. */  
    MB2D      mb2d[1];                 /* Array of button definitions. */  
} MB2INFO;
```

```
typedef MB2INFO *PMB2INFO;
```

MB2INFO Field - cb

cb ([ULONG](#))
Current size of the structure.

MB2INFO Field - hIcon

hIcon ([HPOINTER](#))
Icon handle.

MB2INFO Field - cButtons

cButtons ([ULONG](#))
Number of buttons.

MB2INFO Field - flStyle

flStyle ([ULONG](#))
Icon style flags.

Possible values are described in the following list:

- MB_APPLMODAL**
Message box is application modal. This is the default case. Its owner is disabled; therefore, do not specify the owner as the parent if this option is used.
- MB_ERROR**
Message box contains a stop sign with a white background.
- MB_ICONASTERISK**
Message box contains a asterisk icon.
- MB_CUSTOMICON**
Message box contains a custom icon specified in *hIcon*.
- MB_ICONEXCLAMATION**
Message box contains a exclamation point icon.
- MB_ICONHAND**
Message box contains a hand icon.

MB_ICONQUERY

Message box contains a question mark in a box.

MB_ICONQUESTION

Message box contains a question mark icon.

MB_INFORMATION

Message box contains a black "i" in a box.

MB_MOVEABLE

Message box is moveable.

The message box is displayed with a title bar and a system menu, showing only the Move, Close, and Task Manager choices, which can be selected either by use of the pointing device or by accelerator keys.

MB_NOICON

Message box does not contain an icon.

MB_NONMODAL

Message box is nonmodal (the program continues after displaying the nonmodal message box).

The message box remains visible until the owner window destroys it. Two notification messages, [WM_MSGBOXINIT](#) and [WM_MSGBOXDISMISS](#), are used to support this non-modality.

MB_SYSTEMMODAL

Message box is system modal.

MB_WARNING

Message box contains a black "!" in a box.

MB2INFO Field - hwndNotify

hwndNotify ([HWND](#))

Owner notification handle.

MB2INFO Field - mb2d[1]

mb2d[1] ([MB2D](#))

Array of button definitions.

MENUITEM

Menu item.

```
typedef struct _MENUITEM {  
    SHORT      iPosition;    /* Position. */  
    USHORT     afStyle;      /* Style. */  
    USHORT     afAttribute;  /* Attribute. */  
    USHORT     id;           /* Identity. */  
    HWND       hwndSubMenu;  /* Submenu. */  
    ULONG      hItem;        /* Item. */  
};
```

```
} MENUITEM;  
  
typedef MENUITEM *PMENUITEM;
```

MENUITEM Field - iPosition

iPosition ([SHORT](#))
Position.

MENUITEM Field - afStyle

afStyle ([USHORT](#))
Style.

MENUITEM Field - afAttribute

afAttribute ([USHORT](#))
Attribute.

MENUITEM Field - id

id ([USHORT](#))
Identity.

MENUITEM Field - hwndSubMenu

hwndSubMenu ([HWND](#))
Submenu.

MENUITEM Field - hItem

hlItem ([ULONG](#))
Item.

MINIRECORDCORE

Structure that contains information for records smaller than those defined by the [RECORDCORE](#) data structure. This data structure is used if the CCS_MINIRECORDCORE style bit is specified when a container is created.

```
typedef struct _MINIRECORDCORE {
    ULONG cb; /* Structure size. */
    ULONG flRecordAttr; /* Attributes of container records. */
    POINTL ptlIcon; /* Record position. */
    struct _MINIRECORDCORE *preccNextRecord; /* Pointer to the next linked record. */
    PSZ pszIcon; /* Record text. */
    HPOINTER hptrIcon; /* Record icon. */
} MINIRECORDCORE;

typedef MINIRECORDCORE *PMINIRECORDCORE;
```

MINIRECORDCORE Field - cb

cb ([ULONG](#))
Structure size.

The size (in bytes) of the MINIRECORDCORE structure.

MINIRECORDCORE Field - flRecordAttr

flRecordAttr ([ULONG](#))
Attributes of container records.

Contains any or all of the following:

CRA_COLLAPSED
Specifies that a record is collapsed.

CRA_CURSORED
Specifies that a record will be drawn with a selection cursor.

CRA_DROPONABLE
Specifies that a record can be a target for direct manipulation.

CRA_EXPANDED
Specifies that a record is expanded.

CRA_FILTERED
Specifies that a record is filtered and, therefore, hidden from view.

CRA_INUSE

Specifies that a record will be drawn with in-use emphasis.

CRA_RECORDREADONLY

Prevents a record from being edited directly.

CRA_SELECTED

Specifies that a record will be drawn with selected-state emphasis.

CRA_TARGET

Specifies that a record will be drawn with target emphasis.

MINIRECORDCORE Field - ptllcon

ptllcon ([POINTL](#))

Record position.

Position of a container record in the icon view.

MINIRECORDCORE Field - preccNextRecord

preccNextRecord (struct [_MINIRECORDCORE](#) *)

Pointer to the next linked record.

MINIRECORDCORE Field - pszlcon

pszlcon ([PSZ](#))

Record text.

Text for the container record.

MINIRECORDCORE Field - hptrlcon

hptrlcon ([HPOINTER](#))

Record icon.

Icon that is displayed for the container record.

MLE_SEARCHDATA

Search structure for multi-line entry field.

```
typedef struct _SEARCH {
    USHORT      cb;           /* Size of structure. */
    PCHAR       pchFind;      /* String to search for. */
    PCHAR       pchReplace;   /* String to replace with. */
    SHORT       cchFind;      /* Length of pchFind string. */
    SHORT       cchReplace;   /* Length of pchReplace string. */
    IPT         iptStart;     /* Point at which to start search, or point where string was found. */
    IPT         iptStop;      /* Point at which to stop search. */
    USHORT      cchFound;     /* Length of string found at iptStart. */
} MLE_SEARCHDATA;

typedef MLE_SEARCHDATA *PMLE_SEARCHDATA;
```

MLE_SEARCHDATA Field - cb

cb (USHORT)
Size of structure.

MLE_SEARCHDATA Field - pchFind

pchFind (PCHAR)
String to search for.

MLE_SEARCHDATA Field - pchReplace

pchReplace (PCHAR)
String to replace with.

MLE_SEARCHDATA Field - cchFind

cchFind (SHORT)
Length of *pchFind* string.

MLE_SEARCHDATA Field - cchReplace

cchReplace ([SHORT](#))
Length of *pchReplace* string.

MLE_SEARCHDATA Field - iptStart

iptStart ([IPT](#))
Point at which to start search, or point where string was found.

Non-negative	Point at which to start search.
Negative	Start search from current cursor location.

MLE_SEARCHDATA Field - iptStop

iptStop ([IPT](#))
Point at which to stop search.

Non-negative	Point at which to stop search.
Negative	Stop search at end of text.

MLE_SEARCHDATA Field - cchFound

cchFound ([USHORT](#))
Length of string found at *iptStart*.

MLECTLDATA

Multi-line entry-field (MLE) control data structure.

```
typedef struct _MLECTLDATA {
    USHORT    cbCtlData;        /* Length of control data in bytes. */
    USHORT    afIEFormat;       /* Import/export format. */
    ULONG     cchText;          /* Text limit. */
    IPT       iptAnchor;        /* Selection anchor point. */
    IPT       iptCursor;        /* Selection cursor point. */
    LONG      cxFormat;         /* Formatting-rectangle width in pels. */
    LONG      cyFormat;         /* Formatting-rectangle height in pels. */
    ULONG     afFormatFlags;    /* Format flags. */
} MLECTLDATA;
```

```
typedef MLECTLDATA *PMLECTLDATA;
```

MLECTLDATA Field - cbCtlData

cbCtlData ([USHORT](#))

Length of control data in bytes.

MLECTLDATA Field - afIEFormat

afIEFormat ([USHORT](#))

Import/export format.

This sets the initial import/export format. Setting this value via control data is considered identical to setting it through the [MLM_FORMAT](#) message. The same constants apply here. The default is MLE_CFTEXT.

MLECTLDATA Field - cchText

cchText ([ULONG](#))

Text limit.

The maximum amount of text allowed in the MLE. This value is interpreted identically to the parameter of [MLM_SETTEXTLIMIT](#). A negative value indicates that the length is considered unbounded.

MLECTLDATA Field - iptAnchor

iptAnchor ([IPT](#))

Selection anchor point.

MLECTLDATA Field - iptCursor

iptCursor ([IPT](#))

Selection cursor point.

The *iptAnchor* and *iptCursor* parameters identify the beginning and ending points, respectively, of the selection. These values may range from 0 through the length of the text. The default is 0,0 and can be indicated by entering 0,0.

MLECTLDATA Field - cxFormat

cxFormat ([LONG](#))

Formatting-rectangle width in pels.

MLECTLDATA Field - cyFormat

cyFormat ([LONG](#))

Formatting-rectangle height in pels.

The *cxFormat* and *cyFormat* parameters identify the dimensions in pels of the formatting rectangle, as can be set by the [MLM_SETFORMATRECT](#) message. These values are considered identical to the two fields in the format rectangle structure referenced in that message, and the interpretation of the values in these fields is governed by the *afFormatFlags* field.

The default is the window size in both dimensions, and can be indicated by 0 values.

MLECTLDATA Field - afFormatFlags

afFormatFlags ([ULONG](#))

Format flags.

These flags govern the interpretation of the *cxFormat* and *cyFormat* fields, just as in the [MLM_SETFORMATRECT](#) message. The flag values defined there are also valid in this field. The default is unlimited in both directions, and is of varying size to match the window size.

MLEMARGSTRUCT

Multi-line entry-field margin information.

```
typedef struct _MLEMARGSTRUCT {
    USHORT    afMargins; /* Margin in which the event occurred. */
    USHORT    usMouMsg;  /* Message identity of the original mouse event. */
    IPT       iptNear;   /* Insertion point nearest to the margin event. */
} MLEMARGSTRUCT;

typedef MLEMARGSTRUCT *PMARGSTRUCT;
```

MLEMARGSTRUCT Field - afMargins

afMargins (USHORT)
Margin in which the event occurred.

The left and right margins are defined as including the corners at the top and bottom, and the top and bottom margins are contained between them. Therefore, the corners are included in the sides.

- MLFMARGIN_LEFT
- MLFMARGIN_RIGHT
- MLFMARGIN_TOP
- MLFMARGIN_BOTTOM

MLEMARGSTRUCT Field - usMouMsg

usMouMsg (USHORT)
Message identity of the original mouse event.

MLEMARGSTRUCT Field - iptNear

iptNear (IPT)
Insertion point nearest to the margin event.

MPARAM

A 4-byte message-dependent parameter structure.

```
typedef VOID *MPARAM;
```

Certain elements of information, placed into the parameters of a message, have data types that do not use all four bytes of this data type. The rules governing these cases are:

- | | |
|--------|---|
| BOOL | The value is contained in the low word and the high word is 0. |
| SHORT | The value is contained in the low word and its sign is extended into the high word. |
| USHORT | The value is contained in the low word and the high word is 0. |
| NULL | The entire four bytes are 0. |

The structure of this data type depends on the message. For details, see the description of the particular message.

MQINFO

Message-queue information structure.

```
typedef struct _MQINFO {
    ULONG    cb;           /* Length of structure. */
    PID      pid;          /* Process identity. */
    TID      tid;          /* Thread identity. */
    ULONG    cmsgs;        /* Message count. */
    PVOID    pReserved;    /* Reserved. */
} MQINFO;

typedef MQINFO *PMQINFO;
```

MQINFO Field - cb

cb (ULONG)
Length of structure.

MQINFO Field - pid

pid (PID)
Process identity.

MQINFO Field - tid

tid (TID)
Thread identity.

MQINFO Field - cmsgs

cmsgs (ULONG)
Message count.

MQINFO Field - pReserved

pReserved (PVOID)
Reserved.

MRESULT

A 4-byte message-dependent reply parameter structure.

```
typedef VOID *MRESULT;
```

Certain elements of information, placed into the parameters of a message, have data types that do not use all four bytes of this data type. The rules governing these cases are:

BOOL	The value is contained in the low word and the high word is 0.
SHORT	The value is contained in the low word and its sign is extended into the high word.
USHORT	The value is contained in the low word and the high word is 0.
NULL	The entire four bytes are 0.

The structure of this data type depends on the message. For details, see the description of the particular message.

NOTEBOOKBUTTON

Notebook button data structure for use with the [BKM_SETNOTEBOOKBUTTONS](#) message.

```
typedef struct _NOTEBOOKBUTTON {  
    PSZ      pszText; /* Button text */  
    ULONG    idButton; /* Button ID */  
    LHANDLE  hImage; /* Handle for button or icon. */  
    LONG     flStyle; /* Button style. */  
} NOTEBOOKBUTTON;  
  
typedef NOTEBOOKBUTTON *PNOTEBOOKBUTTON;
```

NOTEBOOKBUTTON Field - pszText

pszText ([PSZ](#))
Button text

The text to be displayed on the button on the notebook page.

NOTEBOOKBUTTON Field - idButton

idButton ([ULONG](#))
Button ID

Button ID. Must be less than BKA_MAXBUTTONID, since IDs greater than this are reserved for system usage.

NOTEBOOKBUTTON Field - hImage

hImage ([LHANDLE](#))
Handle for button or icon.

NOTEBOOKBUTTON Field - flStyle

flStyle ([LONG](#))
Button style.

NOTIFYDELTA

Structure that contains information about the placement of delta information for a container. This structure is used in the [CN_QUERYDELTA](#) container notification code only.

```
typedef struct _NOTIFYDELTA {  
    HWND      hwndCnr; /* Container control handle. */  
    ULONG      fDelta; /* Placement of delta information. */  
} NOTIFYDELTA;  
  
typedef NOTIFYDELTA *PNOTIFYDELTA;
```

NOTIFYDELTA Field - hwndCnr

hwndCnr ([HWND](#))
Container control handle.

NOTIFYDELTA Field - fDelta

fDelta ([ULONG](#))
Placement of delta information. The values can be:

- CMA_DELTATOP
The record that represents the delta value scrolls into view at the top of the client area.
- CMA_DELTABOT
The record that represents the delta value scrolls into view at the bottom of the client area.
- CMA_DELTAHOME
The container scrolls to the beginning of the list of all container records that are available to be inserted into the container, such as the first record in a database.
- CMA_DELTAEND

The container scrolls to the end of the list of all container records that are available to be inserted into the container, such as the last record in a database.

NOTIFYRECORDEMPHASIS

Structure that contains information about emphasis that is being applied to a container record. This structure is used in the [CN_EMPHASIS](#) container notification code only.

```
typedef struct _NOTIFYRECORDEMPHASIS {
    HWND      hwndCnr;          /* Container control handle. */
    RECORDCORE pRecord;         /* Pointer to a RECORDCORE data structure whose emphasis attribute has been
    ULONG      fEmphasisMask;   /* Changed emphasis attributes. */
} NOTIFYRECORDEMPHASIS;

typedef NOTIFYRECORDEMPHASIS *PNOTIFYRECORDEMPHASIS;
```

NOTIFYRECORDEMPHASIS Field - hwndCnr

hwndCnr ([HWND](#))
Container control handle.

NOTIFYRECORDEMPHASIS Field - pRecord

pRecord ([RECORDCORE](#))
Pointer to a [RECORDCORE](#) data structure whose emphasis attribute has been changed.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of RECORDCORE in all applicable data structures and messages.

NOTIFYRECORDEMPHASIS Field - fEmphasisMask

fEmphasisMask ([ULONG](#))
Changed emphasis attributes.

Specifies the emphasis attribute or attributes that changed in the container record. The following states can be combined with a logical OR operator (|):

- CRA_CURSORED
- CRA_INUSE
- CRA_SELECTED.

NOTIFYRECORDENTER

Structure that contains information about the input device that is being used with the container control. This structure is used in the [CN_ENTER](#) container notification code only.

```
typedef struct _NOTIFYRECORDENTER {  
    HWND      hwndCnr; /* Container control handle. */  
    ULONG     fKey;    /* Flag. */  
    PRECORDCORE pRecord; /* Pointer to the RECORDCORE data structure over which an action occurred. */  
} NOTIFYRECORDENTER;  
  
typedef NOTIFYRECORDENTER *PNOTIFYRECORDENTER;
```

NOTIFYRECORDENTER Field - hwndCnr

hwndCnr ([HWND](#))
Container control handle.

NOTIFYRECORDENTER Field - fKey

fKey ([ULONG](#))
Flag.

Flag that determines whether the Enter key was pressed or the select button was double-clicked.

TRUE	The Enter key was pressed.
FALSE	The select button was double-clicked.

NOTIFYRECORDENTER Field - pRecord

pRecord ([PRECORDCORE](#))
Pointer to the [RECORDCORE](#) data structure over which an action occurred.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

- If a user presses the Enter key, a pointer to the record with the selection cursor is returned.
- If a user double-clicks the select button when the pointer of the pointing device is over a record, a pointer to the record is returned.

- If a user double-clicks the select button when the pointer of the pointing device is over white space, NULL is returned.

NOTIFYSCROLL

Structure that contains information about scrolling a container control window. This structure is used in the [CN_SCROLL](#) container notification code only.

```
typedef struct _NOTIFYSCROLL {
    HWND     hwndCnr;    /* Container control handle. */
    LONG     lScrollInc; /* Scroll amount. */
    ULONG    fScroll;    /* Scroll flags. */
} NOTIFYSCROLL;

typedef NOTIFYSCROLL *PNOTIFYSCROLL;
```

NOTIFYSCROLL Field - hwndCnr

hwndCnr ([HWND](#))
Container control handle.

NOTIFYSCROLL Field - lScrollInc

lScrollInc ([LONG](#))
Scroll amount.

Amount (in pixels) by which the window scrolled.

NOTIFYSCROLL Field - fScroll

fScroll ([ULONG](#))
Scroll flags.

Flags that show the direction in which the window scrolled and the window that was scrolled.

CMA_HORIZONTAL

A window was scrolled horizontally. If the split details view window is scrolled, a logical OR operator (|) is used to combine the CMA_HORIZONTAL attribute with either the CMA_LEFT attribute or the CMA_RIGHT attribute to indicate which window was scrolled. If the unsplit details view window is scrolled, the CMA_HORIZONTAL attribute is combined with the CMA_LEFT attribute.

CMA_VERTICAL

The container window scrolled vertically. If the split details view window is scrolled, a logical OR operator (|) is used to combine the CMA_VERTICAL attribute with the CMA_LEFT attribute and the CMA_RIGHT attribute. If the unsplit details view window is scrolled, the CMA_VERTICAL attribute is combined with the CMA_LEFT attribute.

OBJCLASS

Object class structure.

```
typedef struct _OBJCLASS {
    struct _OBJCLASS *pNext; /* Pointer to the next object class structure. */
    PSZ pszClassName; /* Class name. */
    PSZ pszModName; /* Module name. */
} OBJCLASS;

typedef OBJCLASS *POBJCLASS;
```

OBJCLASS Field - pNext

pNext (struct _OBJCLASS *)
Pointer to the next object class structure.

OBJCLASS Field - pszClassName

pszClassName (PSZ)
Class name.

OBJCLASS Field - pszModName

pszModName (PSZ)
Module name.

MLEOVERFLOW

Overflow error structure for multiline entry field.

```
typedef struct _MLEOVERFLOW {
    ULONG afErrInd; /* One or more EFR_* flags. */
    LONG nBytesOver; /* Number of bytes over the limit. */
    LONG pixHorzOver; /* Number of pels over the horizontal limit. */
    LONG pixVertOver; /* Number of pels over the vertical limit. */
} MLEOVERFLOW;

typedef MLEOVERFLOW *POVERFLOW;
```

MLEOVERFLOW Field - afErrInd

afErrInd ([ULONG](#))

One or more EFR_* flags.

MLEOVERFLOW Field - nBytesOver

nBytesOver ([LONG](#))

Number of bytes over the limit.

MLEOVERFLOW Field - pixHorzOver

pixHorzOver ([LONG](#))

Number of pels over the horizontal limit.

MLEOVERFLOW Field - pixVertOver

pixVertOver ([LONG](#))

Number of pels over the vertical limit.

OWNERBACKGROUND

Structure that contains information about painting the container window's background by the container owner. This structure is used in the [CM_PAINTBACKGROUND](#) container message only.

```
typedef struct _OWNERBACKGROUND {
    HWND    hwnd;           /* Window handle. */
    HPS      hps;           /* Presentation-space handle. */
    RECTL    rclBackground; /* Background rectangle. */
    LONG     idWindow;       /* Window ID. */
} OWNERBACKGROUND;

typedef OWNERBACKGROUND *POWNERBACKGROUND;
```

OWNERBACKGROUND Field - hwnd

hwnd (**HWND**)
Window handle.

Handle of the window to be painted.

OWNERBACKGROUND Field - hps

hps (**HPS**)
Presentation-space handle.

OWNERBACKGROUND Field - rclBackground

rclBackground (**RECTL**)
Background rectangle.

Background rectangle in window coordinates.

OWNERBACKGROUND Field - idWindow

idWindow (**LONG**)
Window ID.

Identity of the window to be painted.

OWNERITEM

Owner item.

```
typedef struct _OWNERITEM {  
    HWND      hwnd;          /* Window handle. */  
    HPS       hps;          /* Presentation-space handle. */  
    ULONG     fsState;       /* State. */  
    ULONG     fsAttribute;   /* Attribute. */  
    ULONG     fsStateOld;    /* Old state. */  
    ULONG     fsAttributeOld; /* Old attribute. */  
    RECTL     rclItem;       /* Item rectangle. */  
    LONG      idItem;        /* Item identity. */  
};
```



```
    ULONG      hItem;          /* Item. */  
} OWNERITEM;  
  
typedef OWNERITEM *POWNERITEM;
```

OWNERITEM Field - hwnd

hwnd ([HWND](#))
Window handle.

OWNERITEM Field - hps

hps ([HPS](#))
Presentation-space handle.

OWNERITEM Field - fsState

fsState ([ULONG](#))
State.

OWNERITEM Field - fsAttribute

fsAttribute ([ULONG](#))
Attribute.

OWNERITEM Field - fsStateOld

fsStateOld ([ULONG](#))
Old state.

OWNERITEM Field - fsAttributeOld

fsAttributeOld (ULONG)
Old attribute.

OWNERITEM Field - rcItem

rcItem (RECTL)
Item rectangle.

OWNERITEM Field - idItem

idItem (LONG)
Item identity.

OWNERITEM Field - hItem

hItem (ULONG)
Item.

PAGEINFO

Settings page information structure.

```
typedef struct _PAGEINFO {  
    ULONG        cb;                /* Length of PAGEINFO structure. */  
    HWND         hwndPage;          /* Handle of page. */  
    PFNWP        pfnwp;             /* Window procedure. */  
    ULONG        resid;             /* Resource identity. */  
    PVOID        pCreateParams;     /* Pointer to creation parameters. */  
    USHORT       dlgid;             /* Dialog identity. */  
    USHORT       usPageStyleFlags;  /* Notebook control-page style flags. */  
    USHORT       usPageInsertFlags; /* Notebook control-page insertion flags. */  
    USHORT       usSettingsFlags;   /* Settings flag. */  
    PSZ          pszName;           /* Pointer to a string containing the page name. */  
    USHORT       idDefaultHelpPanel; /* Identity of the default help panel. */  
    USHORT       usReserved2;        /* Reserved value; must be zero. */  
    PSZ          pszHelpLibraryName; /* Pointer to the name of the help file. */  
    PUSHORT      pHelpSubtable;      /* Pointer to the help subtable. */  
    HMODULE       hmodHelpSubtable;  /* Module handle for the help subtable. */  
    ULONG        ulPageInsertId;    /* Notebook control-page identity. */  
} PAGEINFO;  
  
typedef PAGEINFO *PPAGEINFO;
```

PAGEINFO Field - cb

cb ([ULONG](#))
Length of PAGEINFO structure.

PAGEINFO Field - hwndPage

hwndPage ([HWND](#))
Handle of page.

PAGEINFO Field - pfnwp

pfnwp ([PFNWP](#))
Window procedure.

PAGEINFO Field - resid

resid ([ULONG](#))
Resource identity.

PAGEINFO Field - pCreateParams

pCreateParams ([PVOID](#))
Pointer to creation parameters.

PAGEINFO Field - dlgid

dlgid ([USHORT](#))

Dialog identity.

PAGEINFO Field - usPageStyleFlags

usPageStyleFlags ([USHORT](#))
Notebook control-page style flags.

PAGEINFO Field - usPageInsertFlags

usPageInsertFlags ([USHORT](#))
Notebook control-page insertion flags.

PAGEINFO Field - usSettingsFlags

usSettingsFlags ([USHORT](#))
Settings flag.

This flag must be set to one of the following values:

0

You will not get page numbers.

SETTINGS_PAGE_NUMBERS

Page numbers will automatically be put on the status line for pages that have minor pages under the major tab page.

If you want to use the page numbers, make sure all pages have this setting.

PAGEINFO Field - pszName

pszName ([PSZ](#))
Pointer to a string containing the page name.

PAGEINFO Field - idDefaultHelpPanel

idDefaultHelpPanel ([USHORT](#))

Identity of the default help panel.

PAGEINFO Field - usReserved2

usReserved2 ([USHORT](#))

Reserved value; must be zero.

PAGEINFO Field - pszHelpLibraryName

pszHelpLibraryName ([PSZ](#))

Pointer to the name of the help file.

PAGEINFO Field - pHelpSubtable

pHelpSubtable ([PUSHORT](#))

Pointer to the help subtable.

PAGEINFO Field - hmodHelpSubtable

hmodHelpSubtable ([HMODULE](#))

Module handle for the help subtable.

PAGEINFO Field - ulPageInsertId

ulPageInsertId ([ULONG](#))

Notebook control-page identity.

PAGESELECTNOTIFY

Structure that contains information about the application page being selected.

```
typedef struct _PAGESELECTNOTIFY {
    HWND      hwndBook;      /* Notebook window handle. */
    ULONG      ulPageIdCur;  /* Current top page identifier. */
    ULONG      ulPageIdNew;   /* New top page identifier. */
} PAGESELECTNOTIFY;

typedef PAGESELECTNOTIFY *PPAGESELECTNOTIFY;
```

PAGESELECTNOTIFY Field - hwndBook

hwndBook ([HWND](#))
Notebook window handle.

PAGESELECTNOTIFY Field - ulPageIdCur

ulPageIdCur ([ULONG](#))
Current top page identifier.

PAGESELECTNOTIFY Field - ulPageIdNew

ulPageIdNew ([ULONG](#))
New top page identifier.

PANOSE

The *Panose* field in the font metrics will allow for quantitative descriptions of the visual properties of font faces. The PANOSE definition contains ten digits, each of which currently describes up to sixteen variations.

```
typedef struct _PANOSE {
    BYTE      bFamilyType;      /* Family kind. */
    BYTE      bSerifStyle;      /* Serif style. */
    BYTE      bWeight;          /* Weight. */
    BYTE      bProportion;      /* Proportion. */
    BYTE      bContrast;        /* Contrast. */
    BYTE      bStrokeVariation; /* Stroke Variation. */
    BYTE      bArmStyle;        /* Arm Style. */
    BYTE      bLetterform;      /* Letterform. */
    BYTE      bMidline;         /* Midline. */
    BYTE      bXHeight;         /* X-Height. */
    BYTE      fbPassedISO;      /* Font passed ISO test. */
    BYTE      fbFailedISO;      /* Font failed ISO test. */
} PANOSE;

typedef PANOSE *PPANOSE;
```

PANOSE Field - bFamilyType

bFamilyType (BYTE)
Family kind.

0	Any
1	No Fit
2	Text and Display
3	Script
4	Decorative
5	Pictorial

PANOSE Field - bSerifStyle

bSerifStyle (BYTE)
Serif style.

0	Any
1	No Fit
2	Cove
3	Obtuse Cove
4	Square Cove
5	Obtuse Square Cove
6	Square
7	Thin
8	Bone
9	Exaggerated
10	Triangle
11	Normal Sans
12	Obtuse Sans
13	Perp Sans
14	Flared
15	Rounded

PANOSE Field - bWeight

bWeight (BYTE)
Weight.

0	Any
1	No Fit
2	Very Light
3	Light
4	Thin
5	Book
6	Medium

7	Demi
8	Bold
9	Heavy
10	Black
11	Nord

PANOSE Field - bProportion

bProportion (BYTE)
Proportion.

0	Any
1	No Fit
2	Old Style
3	Modern
4	Even Width
5	Expanded
6	Condensed
7	Very Expanded
8	Very Condensed
9	Monospaced

PANOSE Field - bContrast

bContrast (BYTE)
Contrast.

0	Any
1	No Fit
2	None
3	Very Low
4	Low
5	Medium Low
6	Medium
7	Medium High
8	High
9	Very High

PANOSE Field - bStrokeVariation

bStrokeVariation (BYTE)
Stroke Variation.

0	Any
1	No Fit
2	Gradual/Diagonal
3	Gradual/Transitional
4	Gradual/Vertical

5	Gradual/Horizontal
6	Rapid/Vertical
7	Rapid/Horizontal
8	Instant/Vertical

PANOSE Field - bArmStyle

bArmStyle (BYTE)

Arm Style.

0	Any
1	No Fit
2	Straight Arms/Horizontal
3	Straight Arms/Wedge
4	Straight Arms/Vertical
5	Straight Arms/Single Serif
6	Straight Arms/Double Serif
7	Non-Straight Arms/Horizontal
8	Non-Straight Arms/Wedge
9	Non-Straight Arms/Vertical
10	Non-Straight Arms/Single Serif
11	Non-Straight Arms/Double Serif

PANOSE Field - bLetterform

bLetterform (BYTE)

Letterform.

0	Any
1	No Fit
2	Normal/Contact
3	ONormal/Weighted
4	ONormal/Boxed
5	ONormal/Flattened
6	ONormal/Rounded
7	ONormal/Off Center
8	ONormal/Square
9	Oblique/Contact
10	Oblique/Weighted
11	Oblique/Boxed
12	Oblique/Flattened
13	Oblique/Rounded
14	Oblique/Off Center
15	Oblique/Square

PANOSE Field - bMidline

bMidline (BYTE)

Midline.

0	Any
1	No Fit
2	Standard/Trimmed
3	Standard/Pointed
4	Standard/Serifed
5	High/Trimmed
6	High/Pointed
7	High/Serifed
8	Constant/Trimmed
9	Constant/Pointed
10	Constant/Serifed
11	Low/Trimmed
12	Low/Pointed
13	Low/Serifed

PANOSE Field - bXHeight

bXHeight (BYTE)
X-Height.

0	Any
1	No Fit
2	Constant/Small
3	Constant/Standard
4	Constant/Large
5	Ducking/Small
6	Ducking/Standard
7	Ducking/Large

PANOSE Field - fbPassedISO

fbPassedISO (BYTE)
Font passed ISO test.

The following flags indicate those displays and resolutions at which the font complied with ISO 9241:

FM_ISO_9518_640
FM_ISO_9515_640
FM_ISO_9515_1024
FM_ISO_9517_640
FM_ISO_9517_1024

PANOSE Field - fbFailedISO

fbFailedISO (BYTE)
Font failed ISO test.

The following flags indicate those displays and resolutions at which the font did not comply with ISO 9241:

FM_ISO_9518_640
FM_ISO_9515_640
FM_ISO_9515_1024
FM_ISO_9517_640
FM_ISO_9517_1024

PARAM

Presentation parameter attribute definition.

```
typedef struct _PARAM {  
    ULONG    id;        /* Attribute type identity. */  
    ULONG    cb;        /* Byte count of the ab parameter. */  
    BYTE     ab[1];     /* Attribute value. */  
} PARAM;  
  
typedef PARAM *PPARAM;
```

PARAM Field - id

id (ULONG)

Attribute type identity.

These identities are in the range of 0x00000000 to 0xFFFFFFFF. The window manager uses values of this parameter in the range 0x00000000 to PP_USER; therefore, an application cannot define private presentation parameter attribute identities in this range. An application should use [WinAddAtom](#) to guarantee obtaining a unique identity.

PP_FOREGROUNDCOLOR
 Foreground color (in [RGB](#)) attribute.
PP_BACKGROUNDCOLOR
 Background color (in [RGB](#)) attribute.
PP_FOREGROUNDINDEX
 Foreground color index attribute.
PP_BACKGROUNDINDEX
 Background color index attribute.
PP_HILITEFOREGROUNDINDEX
 Highlighted foreground color (in [RGB](#)) attribute, for example for selected menu items.
PP_HILITEBACKGROUNDINDEX
 Highlighted background color (in [RGB](#)) attribute.
PP_HILITEFOREGROUNDINDEX
 Highlighted foreground color index attribute.
PP_HILITEBACKGROUNDINDEX
 Highlighted background color index attribute.
PP_DISABLEDFOREGROUNDINDEX
 Disabled foreground color (in [RGB](#)) attribute.
PP_DISABLEDBACKGROUNDINDEX
 Disabled background color (in [RGB](#)) attribute.
PP_DISABLEDFOREGROUNDINDEX
 Disabled foreground color index attribute.
PP_DISABLEDBACKGROUNDINDEX
 Disabled background color index attribute.
PP_BORDERINDEX
 Border color (in [RGB](#)) attribute.
PP_BORDERINDEX
 Border color index attribute.
PP_FONTNAMEINDEX
 Font name and size attribute.
PP_FONTHANDLE

Font handle attribute.
PP_FONTNAMESIZE
Reserved attribute.
PP_ACTIVECOLOR
Active color value of data type [RGB](#).
PP_ACTIVECOLORINDEX
Active color index value of data type [LONG](#).
PP_INACTIVECOLOR
Inactive color value of data type [RGB](#).
PP_INACTIVECOLORINDEX
Inactive color index value of data type [LONG](#).
PP_ACTIVETEXTFGNDCOLOR
Active text foreground color value of data type [RGB](#).
PP_ACTIVETEXTFGNDCOLORINDEX
Active text foreground color index value of data type [LONG](#).
PP_ACTIVETEXTBGNDCOLOR
Active text background color value of data type [RGB](#).
PP_ACTIVETEXTBGNDCOLORINDEX
Active text background color index value of data type [LONG](#).
PP_INACTIVETEXTFGNDCOLOR
Inactive text foreground color value of data type [RGB](#).
PP_INACTIVETEXTFGNDCOLORINDEX
Inactive text foreground color index value of data type [LONG](#).
PP_INACTIVETEXTBGNDCOLOR
Inactive text background color value of data type [RGB](#).
PP_INACTIVETEXTBGNDCOLORINDEX
Inactive text background color index value of data type [LONG](#).
PP_SHADOW
Changes the color used for drop shadows on certain controls.
PP_MENUFOREGROUNDColor
Menu foreground color.
PP_MENUFOREGROUNDColorINDEX
Menu foreground color index.
PP_MENUBACKGROUNDColor
Menu background color.
PP_MENUBACKGROUNDColorINDEX
Menu background color index.
PP_MENUHILITEFGNDCOLOR
Menu highlighted foreground color.
PP_MENUHILITEFGNDCOLORINDEX
Menu highlighted foreground color index.
PP_MENUHILITEBGNDColor
Menu highlighted background color.
PP_MENUHILITEBGNDColorINDEX
Menu highlighted background color index.
PP_MENUDISABLEDFGNDCOLOR
Menu disabled foreground color.
PP_MENUDISABLEDFGNDCOLORINDEX
Menu disabled foreground color index.
PP_MENUDISABLEDBGNDColor
Menu disabled background color.
PP_MENUDISABLEDBGNDColorINDEX
Menu disabled background color index.
PP_SHADOWTEXTColor
Shadow text color.
PP_SHADOWTEXTColorINDEX
Shadow text color index.
PP_SHADOWHILITEFGNDCOLOR
Shadow highlighted foreground color.
PP_SHADOWHILITEFGNDCOLORINDEX
Shadow highlighted foreground color index.
PP_SHADOWHILITEBGNDColor
Shadow highlighted background color.
PP_SHADOWHILITEBGNDColorINDEX
Shadow highlighted background color index.
PP_ICONTEXTBACKGROUNDColor
Icon text background color.
PP_ICONTEXTBACKGROUNDColorINDEX
Icon text background color index.
PP_BORDERLIGHTColor
Border light color.
PP_BORDERDARKColor
Border dark color.

PP_BORDER2COLOR
Second border color.
PP_BORDER2LIGHTCOLOR
Second border light color.
PP_BORDER2DARKCOLOR
Second border dark color.
PP_BORDERDEFAULTCOLOR
Border default color.
PP_FIELDBACKGROUNDColor
Field background color.
PP_BUTTONBACKGROUNDColor
Button background color.
PP_BUTTONBORDERLIGHTColor
Button border light color.
PP_BUTTONBORDERDARKColor
Button border dark color.
PP_ARROWColor
Arrow color.
PP_ARROWBORDERLIGHTColor
Arrow border light color.
PP_ARROWBORDERDARKColor
Arrow border dark color.
PP_ARROWDISABLEDCOLOR
Arrow disabled color.
PP_CHECKLIGHTColor
Check light color.
PP_CHECKMIDDLEColor
Check middle color.
PP_CHECKDARKColor
Check dark color.
PP_PAGEFOREGROUNDCOLOR
Page foreground color.
PP_PAGEBACKGROUNDColor
Page background color.
PP_MAJORTABFOREGROUNDCOLOR
Major tab foreground color.
PP_MAJORTABBACKGROUNDColor
Major tab background color.
PP_MINORTABFOREGROUNDCOLOR
Minor tab foreground color.
PP_MINORTABBACKGROUNDColor
Minor tab background color.
PP_USER
This is a user-defined presentation parameter.

PARAM Field - cb

cb (ULONG)
Byte count of the *ab* parameter.

PARAM Field - ab[1]

ab[1] (BYTE)
Attribute value.

This value is the format of the attribute type identity that was specified.

PCH

Pointer to a character string.

```
typedef unsigned char *PCH;
```

PCSZ

Pointer to a constant null-terminated string.

```
typedef const char *PCSZ;
```

PDEVOPENDATA

Open device-data array.

This data type points to data whose format is described by the [DEVOPENSTRUC](#) data type.

```
typedef PSZ *PDEVOPENDATA;
```

PFN

Pointer to a procedure.

```
typedef _PFN *PFN;
```

In the header file, this is a two-part definition as shown below:

```
typedef int (APIENTRY _PFN) ();  
typedef _PFN *PFN;
```

PFNWP

Pointer to a window procedure.

This is the standard function definition for window procedures.

```
typedef FNWP *PFNWP;
```

The first argument ([HWND](#)) is the handle of the window receiving the message. The second argument ([ULONG](#)) is a message identifier. The third argument ([MPARAM](#)) is the first message parameter (mp1). The fourth argument ([MPARAM](#)) is the second message parameter (mp2). The function returns an [MRESULT](#). Each message has a specific set of possible return codes. The window procedure must return a value that is appropriate for the message being processed.

In the header file, this is a two-part definition as shown below:

```
typedef MRESULT (EXPENTRY FNWP)(HWND, ULONG, MPARAM, MPARAM);
typedef FNWP *PFNWP;
```

Window procedures must be EXPORTED in the definitions file used by the linker.

PID

Process identity.

```
typedef LHANDLE PID;
```

PIX

Pel count for multi-line entry field.

```
typedef LONG PIX;
```

PKERNPAIRS

Pointer to KERNPAIRS data structure.

```
typedef KERNPAIRS *PKERNPAIRS;
```

POINTERINFO

Pointer-information structure.

```
typedef struct _POINTERINFO {
    ULONG      fPointer;      /* Bit-map size indicator. */
    LONG       xHotSpot;      /* X-coordinate of action point. */
    LONG       yHotSpot;      /* Y-coordinate of action point. */
    HBITMAP    hbmPointer;     /* Bit-map handle of pointer. */
    HBITMAP    hbmColor;       /* Bit-map handle of color bit map. */
    HBITMAP    hbmMiniPointer; /* Bit-map handle of a pointer to a mini bit map. */
    HBITMAP    hbmMiniColor;   /* Bit-map handle of mini color bit map. */
}
```

```
} POINTERINFO;  
  
typedef POINTERINFO *PPOINTERINFO;
```

POINTERINFO Field - fPointer

fPointer ([ULONG](#))
Bit-map size indicator.

TRUE	Pointer-sized bit map
FALSE	Icon-sized bit map.

POINTERINFO Field - xHotSpot

xHotSpot ([LONG](#))
X-coordinate of action point.

POINTERINFO Field - yHotSpot

yHotSpot ([LONG](#))
Y-coordinate of action point.

POINTERINFO Field - hbmPointer

hbmPointer ([HBITMAP](#))
Bit-map handle of pointer.

POINTERINFO Field - hbmColor

hbmColor ([HBITMAP](#))
Bit-map handle of color bit map.

POINTERINFO Field - hbmMiniPointer

hbmMiniPointer ([HBITMAP](#))
Bit-map handle of a pointer to a mini bit map.

POINTERINFO Field - hbmMiniColor

hbmMiniColor ([HBITMAP](#))
Bit-map handle of mini color bit map.

POINTL

Point structure (long integers).

```
typedef struct _POINTL {
    LONG    x; /* X-coordinate. */
    LONG    y; /* Y-coordinate. */
} POINTL;

typedef POINTL *PPOINTL;
```

POINTL Field - x

x ([LONG](#))
X-coordinate.

POINTL Field - y

y ([LONG](#))
Y-coordinate.

POINTS

Point structure (short integers).

```
typedef struct _POINTS {
    SHORT      x; /* X-coordinate. */
    SHORT      y; /* Y-coordinate. */
} POINTS;

typedef POINTS *PPOINTS;
```

POINTS Field - x

x (**SHORT**)
X-coordinate.

POINTS Field - y

y (**SHORT**)
Y-coordinate.

PORTFROMQ

Structure returned by [SplGetPortFromQ](#).

```
typedef struct _PORTFROMQ {
    PSZ      pszComputer; /* Name of the computer on which the port exists. */
    PSZ      pszPort;     /* Name of the port connected to the queue. */
    PSZ      pszDeviceID; /* Device ID string returned by the printer. */
    ULONG    flBidiCapabilities; /* Printer capabilities. */
    ULONG    flJobs; /* Job control flags. */
} PORTFROMQ;

typedef PORTFROMQ *PPORTFROMQ;
```

PORTFROMQ Field - pszComputer

pszComputer (**PSZ**)
Name of the computer on which the port exists.

This information is useful if the print queue references a network print queue.

PORTFROMQ Field - pszPort

pszPort ([PSZ](#))
Name of the port connected to the queue.

PORTFROMQ Field - pszDeviceID

pszDeviceID ([PSZ](#))
Device ID string returned by the printer.

PORTFROMQ Field - flBidiCapabilities

flBidiCapabilities ([ULONG](#))
Printer capabilities.

See for definitions of the PRTPORT_CAPS_xx defines.

PORTFROMQ Field - flJobs

flJobs ([ULONG](#))
Job control flags.

See for definitions of the PRTSW_JOB_xx defines.

PQMOPENDATA

Open queue-manager data array.

This data type points to data whose format is described by the [DEVOPENSTRUC](#) data type.

`typedef PSZ *PQMOPENDATA;`

PRDINFO3

Print device information structure (level 3).

```
typedef struct _PRDINFO3 {
    PSZ      pszPrinterName; /* Print device name. */
    PSZ      pszUserName;    /* User who submitted job. */
    PSZ      pszLogAddr;     /* Logical address (for example LPT1). */
    USHORT   uJobId;         /* Identity of current job. */
    USHORT   fsStatus;       /* Print destination status. */
    PSZ      pszStatus;      /* Print device comment while printing. */
    PSZ      pszComment;     /* Print device description. */
    PSZ      pszDrivers;     /* Drivers supported by print device. */
    USHORT   time;           /* Time job has been printing (minutes) */
    USHORT   usTimeOut;      /* Device timeout (seconds) */
} PRDINFO3;

typedef PRDINFO3 *PPRDINFO3;
```

PRDINFO3 Field - pszPrinterName

pszPrinterName ([PSZ](#))
Print device name.

PRDINFO3 Field - pszUserName

pszUserName ([PSZ](#))
User who submitted job.

This parameter is valid only while the job is printing. It is NULL for a job submitted locally.

PRDINFO3 Field - pszLogAddr

pszLogAddr ([PSZ](#))
Logical address (for example LPT1).

If NULL or an empty string, the printer is not connected to a logical address.

PRDINFO3 Field - uJobId

uJobId ([USHORT](#))
Identity of current job.

If 0, no job is printing.

PRDINFO3 Field - fsStatus

fsStatus (USHORT)

Print destination status.

Use the mask PRD_STATUS_MASK to determine the print job status:

PRD_ACTIVE	Processing
PRD_PAUSED	Not processing, or paused.

Use the mask PRJ_DEVSTATUS for further information about print job status:

PRJ_COMPLETE	Job complete
PRJ_INTERV	Intervention required
PRJ_ERROR	Error occurred (in this case, <i>pszStatus</i> may contain a comment about the error)
PRJ_DESTOFFLINE	Print device offline
PRJ_DESTPAUSED	Print device paused
PRJ_NOTIFY	Raise alert
PRJ_DESTNOPAPER	Print device out of paper.

PRDINFO3 Field - pszStatus

pszStatus (PSZ)

Print device comment while printing.

A comment posted by the print processor of the print device. This parameter is valid only during printing.

PRDINFO3 Field - pszComment

pszComment (PSZ)

Print device description.

PRDINFO3 Field - pszDrivers

pszDrivers (PSZ)

Drivers supported by print device.

List items are separated by commas. Each printer driver name may have a device name separated by a dot (for example, PLOTTER.HP7475A). The default printer is listed first.

PRDINFO3 Field - time

time (USHORT)

Time job has been printing (minutes)

This parameter applies only during printing.

PRDINFO3 Field - usTimeOut

usTimeOut (USHORT)

Device timeout (seconds)

The time that elapses before the device driver notifies the spooler that the print device has not responded.

PRDRIVINFO

Printer driver information structure (level 0).

```
typedef struct _PRDRIVINFO {  
    CHAR        szDrvName[DRIV_NAME_SIZE+1+DRIV_DEVICENAME_SIZE+1]; /* Name of printer driver. */  
} PRDRIVINFO;  
  
typedef PRDRIVINFO *PPRDRIVINFO;
```

PRDRIVINFO Field - szDrvName[DRIV_NAME_SIZE+1+DRIV_DEVICENAME_SIZE+1]

szDrvName[DRIV_NAME_SIZE+1+DRIV_DEVICENAME_SIZE+1] (CHAR)

Name of printer driver.

This is the name of the printer driver and device is the format of DRIVER.DEVICE. For example "IBM4019.IBM Laserprinter E".

PRESPARAMS

Presentation parameter data.

```
typedef struct _PRESPARAMS {  
    ULONG        cb; /* Length of the aparam parameter, in bytes. */  
    PARAM        aparam[1]; /* Array of presentation attribute parameters. */  
} PRESPARAMS;
```

```
typedef PRESPARAMS *PPRESPARAMS;
```

PRESPARAMS Field - cb

cb ([ULONG](#))
Length of the *aparam* parameter, in bytes.

PRESPARAMS Field - aparam[1]

aparam[1] ([PARAM](#))
Array of presentation attribute parameters.

PRFHOOKPARMS

This strucutre is used for the HK_PLIST_ENTRY hook (see [ProgramListEntryHook](#)), and the HK_PLIST_EXIT hook (see [ProgramListExitHook](#)).

```
typedef struct _PRFHOOKPARMS {  
    COUNT4B    numbytes;  
    ULONG      api;  
    STORAGE    apiparms;  
} PRFHOOKPARMS;  
  
typedef PRFHOOKPARMS *PRFHOOKPARMS;
```

PRFHOOKPARMS Field - numbytes

numbytes ([COUNT4B](#))
Length of API packet structure.

PRFHOOKPARMS Field - api

api ([ULONG](#))
API call identifier.

PLSTAPI_PRFQUERYPROGRAMTITLES

	PrfQueryProgramTitles, WinQueryProgramTitles
PLSTAPI_PRFADDPGRAM	PrfAddProgram, WinAddProgram
PLSTAPI_PRFCHANGEPROGRAM	PrfChangeProgram.
PLSTAPI_PRFQUERYDEFINITION	PrfQueryDefinition, WinQueryDefinition
PLSTAPI_REMOVEPROGRAM	PrfRemoveProgram
PLSTAPI_PRFQUERYPROGRAMHANDLE	PrfQueryProgramHandle
PLSTAPI_PRFCREATEGROUP	PrfCreateGroup, WinCreateGroup
PLSTAPI_PRFDESTROYGROUP	PrfDestroyGroup
PLSTAPI_PRFQUERYPROGRAMTYPE	PrfQueryProgramCategory
INIAPI_PRFWRITEPROFILESTRING	PrfWriteProfileString, WinWriteProfileString
INIAPI_PRFQUERYPROFILESIZE	PrfQueryProfileSize, WinQueryProfileSize
INIAPI_PRFQUERYPROFILEDATA	PrfQueryProfileData, WinQueryProfileData
INIAPI_PRFQUERYPROFILEINT	PrfQueryProfileInt, WinQueryProfileInt
INIAPI_PRFWRITEPROFILEDATA	PrfWriteProfileData, WinWriteProfileData
INIAPI_PRFOPENPROFILE	PrfOpenProfile
INIAPI_PRFCLOSEPROFILE	PrfCloseProfile
INIAPI_PRFRESET	PrfReset
INIAPI_PRFQUERYPROFILE	PrfQueryProfile
INIAPI_PRFQUERYPROFILESTRING	PrfQueryProfileString, WinQueryProfileString

PRFHOOKPARMS Field - apiparms

apiparms (STORAGE)

API parameters packet. The descriptions of these data types follow that of PRFHOOKPARMS.

PLSTAPI_PRFQUERYPROGRAMTITLES	PrfQueryProgramTitles, WinQueryProgramTitles
PLSTAPI_PRFADDPGRAM	PrfAddProgram, WinAddProgram
PLSTAPI_PRFCHANGEPROGRAM	PrfChangeProgram.
PLSTAPI_PRFQUERYDEFINITION	PrfQueryDefinition, WinQueryDefinition
PLSTAPI_REMOVEPROGRAM	PrfRemoveProgram
PLSTAPI_PRFQUERYPROGRAMHANDLE	PrfQueryProgramHandle
PLSTAPI_PRFCREATEGROUP	PrfCreateGroup, WinCreateGroup
PLSTAPI_PRFDESTROYGROUP	PrfDestroyGroup
PLSTAPI_PRFQUERYPROGRAMTYPE	PrfQueryProgramCategory
INIAPI_PRFWRITEPROFILESTRING	PrfWriteProfileString, WinWriteProfileString
INIAPI_PRFQUERYPROFILESIZE	PrfQueryProfileSize, WinQueryProfileSize


```
INIAPI_PRFQUERYPROFILEDATA
    PrfQueryProfileData, WinQueryProfileData
INIAPI_PRFQUERYPROFILEINT
    PrfQueryProfileInt, WinQueryProfileInt
INIAPI_PRFWRITEPROFILEDATA
    PrfWriteProfileData, WinWriteProfileData
INIAPI_PRFOPENPROFILE
    PrfOpenProfile
INIAPI_PRFCLOSEPROFILE
    PrfCloseProfile
INIAPI_PRFRESET
    PrfReset
INIAPI_PRFQUERYPROFILE
    PrfQueryProfile
INIAPI_PRFQUERYPROFILESTRING
    PrfQueryProfileString, WinQueryProfileString
```

PRFPROFILE

Profile structure.

```
typedef struct _PRFPROFILE {
    ULONG    cchUserName; /* Length of user profile name. */
    PSZ      pszUserName; /* User profile name. */
    ULONG    cchSysName; /* Length of system profile name. */
    PSZ      pszSysName; /* System profile name. */
} PRFPROFILE;

typedef PRFPROFILE *PPRFPROFILE;
```

PRFPROFILE Field - cchUserName

cchUserName (ULONG)
Length of user profile name.

PRFPROFILE Field - pszUserName

pszUserName (PSZ)
User profile name.

PRFPROFILE Field - cchSysName

cchSysName (ULONG)
Length of system profile name.

PRFPROFILE Field - pszSysName

pszSysName ([PSZ](#))
System profile name.

PRINTDEST

PRINTDEST data structure.

Contains all the parameters required to issue [DevPostDeviceModes](#) and [DevOpenDC](#) function calls.

```
typedef struct _PRINTDEST {  
    ULONG      cb;           /* Length of data structure, in bytes. */  
    LONG       lType;        /* Type of device context. */  
    PSZ        pszToken;     /* Device-information token. */  
    LONG       lCount;       /* Number of items. */  
    PDEVOPENDATA pdopData;   /* Open device context data area. */  
    ULONG      fl;           /* Flags. */  
    PSZ        pszPrinter;   /* Name of the printer. */  
} PRINTDEST;  
  
typedef PRINTDEST *PPRINTDEST;
```

PRINTDEST Field - cb

cb ([ULONG](#))
Length of data structure, in bytes.

PRINTDEST Field - lType

lType ([LONG](#))
Type of device context.

OD_QUEUED	The device context is queued.
OD_DIRECT	The device context is direct.

PRINTDEST Field - pszToken

pszToken ([PSZ](#))
Device-information token.

This is always `"*"`.

PRINTDEST Field - ICount

ICount ([LONG](#))
Number of items.

This is the number of items present in the *pdopData* field.

PRINTDEST Field - pdopData

pdopData ([PDEVOPENDATA](#))
Open device context data area.

See [DEVOPENSTRUC](#) for information on the format of *pdopData*.

PRINTDEST Field - fl

fl ([ULONG](#))
Flags.

PD_JOB_PROPERTY
This flag indicates that [DevPostDeviceModes](#) should be called with `DPDM_POSTJOBPROP` before calling [DevOpenDC](#).

PRINTDEST Field - pszPrinter

pszPrinter ([PSZ](#))
Name of the printer.

A name that specifies the device; for example, "PRINTER1". The name is used for calling [DevPostDeviceModes](#).

The printer device name can be found by calling [SplQueryQueue](#) and passing to it the information found in the *pszLogAddress* field of the [DEVOPENSTRUC](#) structure pointed to by *pdopData*. [SplQueryQueue](#) returns a [PRQINFO3](#) structure. The *pszPrinters* field in [PRQINFO3](#) contains the printer device name to be used.

PRINTERINFO

Print destination information structure.

This structure is used at information level 0.

```
typedef struct _PRINTERINFO {
    ULONG      flType;          /* Type of printer. */
    PSZ        pszComputerName; /* Computer name. */
    PSZ        pszPrintDestinationName; /* Name of Print Destination. */
    PSZ        pszDescription; /* Description of print destination. */
    PSZ        pszLocalName;    /* Local name of remote print destination. */
} PRINTERINFO;

typedef PRINTERINFO *PPRINTERINFO;
```

PRINTERINFO Field - flType

flType (ULONG)

Type of printer.

This is a flag used to describe the type of print destination:

SPL_PR_QUEUE
Print destination is a queue
SPL_PR_DIRECT_DEVICE
Print destination is a direct print device
SPL_PR_QUEUED_DEVICE
Print destination is a queued print device

PRINTERINFO Field - pszComputerName

pszComputerName (PSZ)

Computer name.

A NULL string specifies the local workstation.

PRINTERINFO Field - pszPrintDestinationName

pszPrintDestinationName (PSZ)

Name of Print Destination.

This is either a queue name or a print device name depending upon the value of *flType*. The maximum length of the name in the network case is 256 (including one byte for the null terminator).

PRINTERINFO Field - pszDescription

pszDescription (PSZ)

Description of print destination.

The maximum length is 48 characters (including one byte for the null terminator).

PRINTERINFO Field - pszLocalName

pszLocalName (PSZ)

Local name of remote print destination.

This is a local port name (for instance "LPT4") that is connected to the remote print destination. A NULL string specifies that no connection exists.

PRJINFO2

Print-job information structure.

This structure provides a subset of the information supplied by [PRJINFO3](#). It minimizes the storage required for job-information retrieval, and is sufficient for most uses.

```
typedef struct _PRJINFO2 {
    USHORT    uJobId;        /* Job identification number. */
    USHORT    uPriority;     /* Job priority. */
    PSZ       pszUserName;   /* User who submitted the job. */
    USHORT    uPosition;    /* Job position in queue. */
    USHORT    fsStatus;     /* Job status. */
    ULONG     ulSubmitted;   /* Time job submitted. */
    ULONG     ulSize;       /* Print-job size (bytes). */
    PSZ       pszComment;    /* Comment string. */
    PSZ       pszDocument;  /* Document name. */
} PRJINFO2;

typedef PRJINFO2 *PPRJINFO2;
```

PRJINFO2 Field - uJobId

uJobId (USHORT)

Job identification number.

PRJINFO2 Field - uPriority

uPriority (USHORT)
Job priority.

The job-priority range is 1 through 99, with 99 the highest job priority. (For queue priorities, 1 is the highest priority.)

The job priority determines the order of jobs in the queue. If multiple queues print to the same printer, the job at the front of each queue is examined. The job with the highest priority is printed first; if there is more than one job with the highest priority, the oldest job with this priority is printed first.

- PRJ_MAX_PRIORITY
Highest priority
- PRJ_MIN_PRIORITY
Lowest priority
- PRJ_NO_PRIORITY
No priority.

PRJINFO2 Field - pszUserName

pszUserName (PSZ)
User who submitted the job.

This parameter applies only to jobs created by a user and enqueued on a remote server. A NULL string signifies a local job.

PRJINFO2 Field - uPosition

uPosition (USHORT)
Job position in queue.

If 1, the job is scheduled to be the next job printed from this queue.

PRJINFO2 Field - fsStatus

fsStatus (USHORT)
Job status.

To find the job status, use the PRJ_QSTATUS mask:

- | | |
|-----------------|---|
| PRJ_QS_QUEUED | Queued |
| PRJ_QS_PAUSED | Paused by a SplHoldJob function |
| PRJ_QS_SPOOLING | Job being created |
| PRJ_QS_PRINTING | Printing (bits 2 through 11 are valid). |

For further information, use the PRJ_DEVSTATUS mask:

- | | |
|-----------------|---------------------------|
| PRJ_COMPLETE | Job complete |
| PRJ_INTERV | Intervention required |
| PRJ_ERROR | Error occurred. |
| PRJ_DESTOFFLINE | Print destination offline |

PRJ_DESTPAUSED	Print destination paused
PRJ_NOTIFY	Alert should be raised
PRJ_DESTNOPAPER	Print destination out of paper
PRJ_DESTFORMCHG	Printer waiting for form change
PRJ_DESTCRTCHG	Printer waiting for cartridge change
PRJ_DESTPENCHG	Printer waiting for pen change.
This bit indicates that the job is deleted:	
PRJ_DELETED	Job deleted.

PRJINFO2 Field - ulSubmitted

ulSubmitted (ULONG)
Time job submitted.

Time format is the same as that stored in the global information segment.

PRJINFO2 Field - ulSize

ulSize (ULONG)
Print-job size (bytes).

PRJINFO2 Field - pszComment

pszComment (PSZ)
Comment string.

Information about the print job. The maximum length of the string is 48 characters (including one byte for the null terminator).

PRJINFO2 Field - pszDocument

pszDocument (PSZ)
Document name.

The document name of the print job (set by the application that submitted the print job). The maximum length of the string is 260 characters.

PRJINFO3

Print-job information structure.

This structure is used when complete job details are required. A subset of this information is supplied by [PRJINFO2](#).

```
typedef struct _PRJINFO3 {
    USHORT      uJobId;           /* Job identification number. */
    USHORT      uPriority;        /* Job priority. */
    PSZ         pszUserName;     /* User who submitted the job. */
    USHORT      uPosition;       /* Job position in queue. */
    USHORT      fsStatus;        /* Job status. */
    ULONG       ulSubmitted;     /* Time job submitted. */
    ULONG       ulSize;          /* Print-job size (bytes). */
    PSZ         pszComment;      /* Comment string. */
    PSZ         pszDocument;     /* Document name. */
    PSZ         pszNotifyName;   /* Messaging alias for print alert. */
    PSZ         pszDataType;     /* Data type of submitted file. */
    PSZ         pszParms;        /* Parameters. */
    PSZ         pszStatus;       /* Status comment. */
    PSZ         pszQueue;        /* Queue name. */
    PSZ         pszQProcName;    /* Queue processor. */
    PSZ         pszQProcParms;   /* Queue processor parameters. */
    PSZ         pszDriverName;   /* Driver name. */
    PDRIVDATA   pDriverData;     /* Job Properties (driver data). */
    PSZ         pszPrinterName;  /* Printer name. */
} PRJINFO3;

typedef PRJINFO3 *PPRJINFO3;
```

PRJINFO3 Field - uJobId

uJobId ([USHORT](#))
Job identification number.

PRJINFO3 Field - uPriority

uPriority ([USHORT](#))
Job priority.

The job-priority range is 1 through 99, with 99 the highest job priority. (For queue priorities, 1 is the highest priority.)

The job priority determines the order of jobs in the queue. If multiple queues print to the same printer, the job on the front of each queue is examined. The job with the highest priority is printed first; if there is more than one job with the highest priority, the oldest job with this priority is printed first.

- PRJ_MAX_PRIORITY
Highest priority
- PRJ_MIN_PRIORITY
Lowest priority
- PRJ_NO_PRIORITY
No priority.

PRJINFO3 Field - pszUserName

pszUserName (PSZ)
User who submitted the job.

This parameter applies only to jobs created by a user on a remote workstation and queued on a server. A NULL string signifies a local job.

PRJINFO3 Field - uPosition

uPosition (USHORT)
Job position in queue.

If 1, the job is scheduled to be the next job printed from this queue.

PRJINFO3 Field - fsStatus

fsStatus (USHORT)
Job status.

To find the job status, use the PRJ_QSTATUS mask:

PRJINFO3 Field - ulSubmitted

ulSubmitted (ULONG)
Time job submitted.

Time format is the same as that stored in the global information segment.

PRJINFO3 Field - ulSize

ulSize (ULONG)
Print-job size (bytes).

PRJINFO3 Field - pszComment

pszComment (PSZ)

Comment string.

Information about the print job.

The maximum length of the string is 48 characters (including one byte for the null terminator).

PRJINFO3 Field - pszDocument

pszDocument (PSZ)

Document name.

The document name of the print job (set by the application that submitted the print job). The maximum length of the string is 260 characters.

PRJINFO3 Field - pszNotifyName

pszNotifyName (PSZ)

Messaging alias for print alert.

This parameter is a computer name and applies only to jobs on a remote server queue. A NULL string is returned for jobs on a local queue.

PRJINFO3 Field - pszDataType

pszDataType (PSZ)

Data type of submitted file.

This is specified by the *pszDataType* parameter in the [DEVOPENSTRUC](#) structure passed to the [DevOpenDC](#) call when the job is created. The name is truncated to fit the field if necessary, and contains a trailing NULL.

PRJINFO3 Field - pszParms

pszParms (PSZ)

Parameters.

The form of this string is:

parm1=val1 parm2=val2 ...

PRJINFO3 Field - pszStatus

pszStatus (PSZ)
Status comment.

A text string, posted by the queue processor, that provides additional job-status information. The default string type is NULL.

PRJINFO3 Field - pszQueue

pszQueue (PSZ)
Queue name.

The name of the queue the job is on.

PRJINFO3 Field - pszQProcName

pszQProcName (PSZ)
Queue processor.

The name of the queue processor.

PRJINFO3 Field - pszQProcParms

pszQProcParms (PSZ)
Queue processor parameters.

Spaces are used to separate parameters.

PRJINFO3 Field - pszDriverName

pszDriverName (PSZ)
Driver name.

The name of the device driver (for example, "LASERJET"). The device name is part of *pDriverData*.

PRJINFO3 Field - pDriverData

pDriverData ([PDRIVDATA](#))

Job Properties (driver data).

The contents are specific to the device driver.

PRJINFO3 Field - pszPrinterName

pszPrinterName ([PSZ](#))

Printer name.

If the job is printing, the printer name, otherwise NULL.

PRJINFO4

Information about the PRJINFO4 data structure, which stores accounting information about print jobs.

```
typedef struct _PRJINFO4 {
    USHORT    uJobID;           /* Spooler job identifier. */
    USHORT    uPriority;        /* Priority of the print job. */
    PSZ       pszUserName;     /* Name of job submitter. */
    USHORT    uPosition;       /* Job position in the print queue. */
    USHORT    fsStatus;        /* Job status bits. */
    ULONG     ulSubmitted;     /* Date and time of job submission. */
    ULONG     ulSize;          /* Size of the spooled job. */
    PSZ       pszComment;      /* Job comment string. */
    PSZ       pszDocument;     /* Job document name. */
    PSZ       pszSpoolFileName; /* Path to spool file if it is not in the standard spool directory. */
    PSZ       pszPortName;     /* Port on which this job is printing. */
    PSZ       pszStatus;       /* Job status string. */
    ULONG     ulPagesSpooled;   /* Total pages spooled so far. */
    ULONG     ulPagesSent;      /* Total pages sent to the printer for this job. */
    ULONG     ulPagesPrinted;   /* Total pages stacked for this job. */
    ULONG     ulTimePrinted;    /* Processing time used in printer. */
    ULONG     ulExtendJobStatus; /* Extended job status flags. */
    ULONG     ulStartPage;     /* First page to begin printing. */
    ULONG     ulEndPage;       /* Last page to print for this job. */
} PRJINFO4;

typedef PRJINFO4 *PPRJINFO4;
```

PRJINFO4 Field - uJobID

uJobID ([USHORT](#))

Spooler job identifier.

PRJINFO4 Field - uPriority

uPriority (USHORT)
Priority of the print job.

PRJINFO4 Field - pszUserName

pszUserName (PSZ)
Name of job submitter.

PRJINFO4 Field - uPosition

uPosition (USHORT)
Job position in the print queue.

PRJINFO4 Field - fsStatus

fsStatus (USHORT)
Job status bits.

PRJINFO4 Field - ulSubmitted

ulSubmitted (ULONG)
Date and time of job submission.

PRJINFO4 Field - ulSize

ulSize (ULONG)
Size of the spooled job.

PRJINFO4 Field - pszComment

pszComment ([PSZ](#))
Job comment string.

PRJINFO4 Field - pszDocument

pszDocument ([PSZ](#))
Job document name.

PRJINFO4 Field - pszSpoolFileName

pszSpoolFileName ([PSZ](#))
Path to spool file if it is not in the standard spool directory.

This field is null except when an application wants to submit a print job without spooling it (due to the size of the print file).
pszSpoolFileName can be set with [SplSetJob](#) using the PRJ_SPOOLFILENAME_PARMNUM(19) parameter.

PRJINFO4 Field - pszPortName

pszPortName ([PSZ](#))
Port on which this job is printing.

PRJINFO4 Field - pszStatus

pszStatus ([PSZ](#))
Job status string.

PRJINFO4 Field - ulPagesSpooled

ulPagesSpooled (ULONG)
Total pages spooled so far.

This value will be 0 (zero) for printer drivers that do not call SplQmNewPage. *ulPagesSpooled* can be set with SplSetJob using the PRJ_PAGESSPOOLED_PARMNUM(20) parameter.

PRJINFO4 Field - ulPagesSent

ulPagesSent (ULONG)
Total pages sent to the printer for this job.

This value reflects the number of pages sent to the printer so far, if PrtNewPage is used by the printer driver. A value of 0 (zero) indicates the printer driver is **not** calling PrtNewPage. *ulPagesSent* can be set with SplSetJob using the PRJ_PAGESSENT_PARMNUM(21) parameter.

PRJINFO4 Field - ulPagesPrinted

ulPagesPrinted (ULONG)
Total pages stacked for this job.

This value will be known only for BIDI printers. It indicates the number of pages the printer has confirmed were output. *ulPagesPrinted* can be set with SplSetJob using the PRJ_PAGESPRINTED_PARMNUM(22) parameter.

PRJINFO4 Field - ulTimePrinted

ulTimePrinted (ULONG)
Processing time used in printer.

If the printer is unable to tell host, 0 (zero) will be used. *ulTimePrinted* can be set with SplSetJob using the PRJ_TIMEPRINTED_PARMNUM(23) parameter.

PRJINFO4 Field - ulExtendJobStatus

ulExtendJobStatus (ULONG)
Extended job status flags.

Values are as follows:

Bit	Description
0	PRJ4_INPRINTER - 0x00000001 Job in printer (0=No; 1=Yes)
1	PRJ4_STACKED - 0x00000002

	Job is stacked (0=No; 1=Yes)
2	PRJ4_HELDINPRINTER - 0x00000004 Job held in printer (0=No; 1=Yes)
3	PRJ4_JOBSTARTED - 0x00000008 Job has begun printing (0=No; 1=Yes)
4 - 31	Reserved

This field can be set with SplSetJob using the PRJ_EXTENDSTATUSus.PARMNUM(24) parameter.

PRJINFO4 Field - ulStartPage

ulStartPage (ULONG)

First page to begin printing.

This value indicates the first page to begin printing, if the printer supports this feature. A value of 0 (zero) indicates printing should begin with the first page. *ulStartPage* can be set with SplSetJob using the PRJ_STARTPAGE_PARMNUM(25) parameter.

PRJINFO4 Field - ulEndPage

ulEndPage (ULONG)

Last page to print for this job.

This value indicates the last page to print for this job, if the printer supports this feature. *ulEndPage* can be set with SplSetJob using the PRJ_ENDPAGE_PARMNUM(26) parameter.

PROGCATEGORY

Program category.

```
typedef ULONG PROGCATEGORY;
```

PROGDETAILS

Program-details structure.

```
typedef struct _PROGDETAILS {
    ULONG      Length;          /* Length of structure. */
    PROGTTYPE  progt;           /* Program type. */
    PSZ        pszTitle;        /* Title. */
    PSZ        pszExecutable;    /* Executable file name (program name). */
    PSZ        pszParameters;    /* Parameter string. */
    PSZ        pszStartupDir;    /* Start-up directory. */
}
```



```
    PSZ        pszIcon;          /* Icon-file name. */
    PSZ        pszEnvironment;   /* Environment string. */
    SWP        swpInitial;       /* Initial window position and size. */
} PROGDETAILS;

typedef PROGDETAILS *PPROGDETAILS;
```

PROGDETAILS Field - Length

Length ([ULONG](#))
Length of structure.

PROGDETAILS Field - progt

progt ([PROGTYPE](#))
Program type.

PROGDETAILS Field - pszTitle

pszTitle ([PSZ](#))
Title.

If pszTitle is NULL, pszExecutable is used.

PROGDETAILS Field - pszExecutable

pszExecutable ([PSZ](#))
Executable file name (program name).

If pszExecutable is NULL, the file specified in the OS2_SHELL statement in the configuration file (CONFIG.SYS) is used. If the OS2_SHELL statement does not exist, the command processor (CMD.EXE for a non-DOS session, or COMMAND.COM for a DOS session) is used.

PROGDETAILS Field - pszParameters

pszParameters ([PSZ](#))

Parameter string.

The pszParameters can be set to NULL.

PROGDETAILS Field - pszStartupDir

pszStartupDir (PSZ)

Start-up directory.

The pszStartupDir can be set to NULL.

PROGDETAILS Field - pszIcon

pszIcon (PSZ)

Icon-file name.

If pszIcon is NULL, the default icon for the executable is used.

PROGDETAILS Field - pszEnvironment

pszEnvironment (PSZ)

Environment string.

A list of null-terminated strings ending with an extra NULL character.

PROGDETAILS Field - swpInitial

swpInitial (SWP)

Initial window position and size.

PROGRAMENTRY

Program-entry structure.

```
typedef struct _PROGRAMENTRY {  
    HPROGRAM    hprog;           /* Program handle. */  
    PROGTTYPE   progt;           /* Program type. */  
    CHAR        szTitle[MAXNAMEL+1]; /* Program title (null-terminated). */  
};
```

```
} PROGRAMENTRY;  
typedef PROGRAMENTRY *PPROGRAMENTRY;
```

PROGRAMENTRY Field - hprog

hprog ([HPROGRAM](#))
Program handle.

PROGRAMENTRY Field - progt

progt ([PROGTYPE](#))
Program type.

PROGRAMENTRY Field - szTitle[MAXNAMEL+1]

szTitle[MAXNAMEL+1] ([CHAR](#))
Program title (null-terminated).

PROGTYPE

Program-type structure.

```
typedef struct _PROGTYPE {  
    PROGCATEGORY    progc;    /* Program category: */  
    ULONG           fbVisible; /* Visibility attribute. */  
} PROGTYPE;  
  
typedef PROGTYPE *PPROGTYPE;
```

PROGTYPE Field - progc

progc ([PROGCATEGORY](#))
Program category:

PROG_DEFAULT

	Default application.
PROG_PM	Presentation Manager application.
PROG_WINDOWABLEVIO	Text-windowed application.
PROG_FULLSCREEN	Full-screen application.
PROG_WINDOWEDVDM	PC DOS executable process (windowed).
PROG_VDM	PC DOS executable process (full screen).
PROG_REAL	PC DOS executable process (full screen). Same as PROG_VDM.
PROG_31_STDSEAMLESSVDM	Windows 3.1 program that will execute in its own windowed WINOS2 session.
PROG_31_STDSEAMLESSCOMMON	Windows 3.1 program that will execute in a common windowed WINOS2 session.
PROG_31_ENHSEAMLESSVDM	Windows 3.1 program that will execute in enhanced compatibility mode in its own windowed WINOS2 session.
PROG_31_ENHSEAMLESSCOMMON	Windows 3.1 program that will execute in enhanced compatibility mode in a common windowed WINOS2 session.
PROG_31_ENH	Windows 3.1 program that will execute in enhanced compatibility mode in a full-screen WINOS2 session.
PROG_31_STD	Windows 3.1 program that will execute in a full-screen WINOS2 session.

PROGTYPE Field - fbVisible

fbVisible (ULONG)

Visibility attribute.

When testing this field, allow for the possibility that other bits may be defined in the future. SHE_INVISIBLE and SHE_PROTECTED can be used to mask the visibility and protected flags, respectively.

SHE_VISIBLE	Visible
SHE_INVISIBLE	Invisible
SHE_UNPROTECTED	Unprotected
SHE_PROTECTED	Protected.

PRPORTINFO

Port information structure (level 0).

```
typedef struct _PRPORTINFO {
    CHAR    szPortName[PDLEN+1]; /* Name of the port. */
} PRPORTINFO;

typedef PRPORTINFO *PPRPORTINFO;
```

PRPORTINFO Field - szPortName[PDLEN+1]

szPortName[PDLEN+1] ([CHAR](#))
Name of the port.

This is the name of the port. For example "LPT1".

PRPORTINFO1

Port information structure (level 1).

```
typedef struct _PRPORTINFO1 {
    PSZ      pszPortName;           /* Name of the port. */
    PSZ      pszPortDriverName;     /* Name of the port driver. */
    PSZ      pszPortDriverPathName; /* Full path name of the port driver. */
} PRPORTINFO1;

typedef PRPORTINFO1 *PPRPORTINFO1;
```

PRPORTINFO1 Field - pszPortName

pszPortName ([PSZ](#))
Name of the port.

This is the name of the port. For example "LPT1".

PRPORTINFO1 Field - pszPortDriverName

pszPortDriverName ([PSZ](#))
Name of the port driver.

This is the name of the port driver. For example "PARALLEL".

PRPORTINFO1 Field - pszPortDriverPathName

pszPortDriverPathName ([PSZ](#))
Full path name of the port driver.

This is the full path name of the port driver. For example "C:\OS2\DLL\PARALLEL.PDR".

PRPORTINFO2

Information about the current port settings, used by [SplQueryPort](#) and [SplSetPort](#).

```
typedef struct _PRPORTINFO2 {  
    PSZ      pszPortName;          /* Name of the port. */  
    PSZ      pszPortDriver;        /* Port driver used by the port. */  
    PSZ      pszProtocolConverter; /* Protocol converter used by the port. */  
    ULONG    ulReserved;           /* Set to 0 (zero) for now. */  
    ULONG    ulMode;               /* BIDI mode of the port. */  
    ULONG    ulPriority;           /* Currently not used. */  
} PRPORTINFO2;  
  
typedef PRPORTINFO2 *PPRPORTINFO2;
```

PRPORTINFO2 Field - pszPortName

pszPortName ([PSZ](#))
Name of the port.

PRPORTINFO2 Field - pszPortDriver

pszPortDriver ([PSZ](#))
Port driver used by the port.

PRPORTINFO2 Field - pszProtocolConverter

pszProtocolConverter ([PSZ](#))
Protocol converter used by the port.

PRPORTINFO2 Field - ulReserved

ulReserved ([ULONG](#))
Set to 0 (zero) for now.

PRPORTINFO2 Field - ulMode

ulMode (ULONG)
BIDI mode of the port.

Values are as follows:

Value	Description
1	PRPORT_AUTODETECT Auto-detect mode. The port driver will tell the spooler if the printer is BIDI-capable. This is the default.
2	PRPORT_DISABLE_BIDI BIDI is disabled. The spooler will not use the printer in BIDI mode. This value overrides the port driver.
3	PRPORT_ENABLE_BIDI BIDI is enabled. This is set only if an application knows a printer is capable of using a protocol converter, but the port driver does not know which converter to use. The application must also supply the protocol converter name when setting a port to this mode.

PRPORTINFO2 Field - ulPriority

ulPriority (ULONG)
Currently not used.

PRQINFO3

Print-queue information structure.

This structure is used at information levels 3 and 4.

```
typedef struct _PRQINFO3 {
    PSZ      pszName;           /* Queue name. */
    USHORT   uPriority;          /* Queue priority. */
    USHORT   uStartTime;        /* Minutes after midnight when queue becomes active. */
    USHORT   uUntilTime;        /* Minutes after midnight. when queue ceases to be active. */
    USHORT   fsType;            /* Queue type. */
    PSZ      pszSepFile;        /* Separator-page file. */
    PSZ      pszPrProc;         /* Default queue-processor. */
    PSZ      pszParms;          /* Queue parameters. */
    PSZ      pszComment;        /* Queue description. */
    USHORT   fsStatus;          /* Queue status. */
    USHORT   cJobs;             /* Number of jobs in queue. */
    PSZ      pszPrinters;        /* Print devices connected to queue. */
    PSZ      pszDriverName;     /* Default device driver. */
    PDRIVDATA pDriverData;      /* Default queue job properties. */
} PRQINFO3;

typedef PRQINFO3 *PPRQINFO3;
```

PRQINFO3 Field - pszName

pszName (PSZ)
Queue name.

The maximum length of the name in the network case is 256 (including one byte for zero termination).

PRQINFO3 Field - uPriority

uPriority (USHORT)
Queue priority.

The range is 1 through 9, with 1 being the highest queue priority.

The default job priority (DefJobPrio) is determined from:
 $\text{DefJobPrio} = 100 - (10 * \text{uPriority})$.

If a job is added with **PRJ_NO_PRIORITY** specified, DefJobPrio is used. If a default priority higher than the default job priority is specified, the default job priority is used. If a default priority lower than the default is specified, the specified job priority is used.

- PRQ_DEF_PRIORITY
Default priority
- PRQ_MAX_PRIORITY
Highest priority
- PRQ_MIN_PRIORITY
Minimum priority
- PRQ_NO_PRIORITY
No priority.

PRQINFO3 Field - uStartTime

uStartTime (USHORT)
Minutes after midnight when queue becomes active.

For example, the value 75 represents 1:15 a.m.

If *uStartTime* and *uUntilTime* are both 0, the print queue is always available.

PRQINFO3 Field - uUntilTime

uUntilTime (USHORT)
Minutes after midnight. when queue ceases to be active.

For example, the value 1200 represents 8 p.m.

If *uUntilTime* and *uStartTime* are both 0, the print queue is always available.

PRQINFO3 Field - fsType

fsType (USHORT)

Queue type.

PRQ3_TYPE_RAW

Data is always enqueued in the device specific format.

PRQ3_TYPE_BYPASS

Allows the spooler to bypass the queue processor and send data directly to the Printer Driver. Setting this bit allows the spooler to print jobs of type PM_Q_RAW while they are still being spooled.

PRQ3_TYPE_APPDEFAULT

This bit is set for the application default queue only.

PRQINFO3 Field - pszSepFile

pszSepFile (PSZ)

Separator-page file.

The path and file name of a separator-page file on the target computer.

This file contains formatting information for the page or pages to be used between print jobs. A relative path name is taken as relative to the current spool directory. A NULL string indicates no separator page.

PRQINFO3 Field - pszPrProc

pszPrProc (PSZ)

Default queue-processor.

PRQINFO3 Field - pszParms

pszParms (PSZ)

Queue parameters.

This can be any text string or a NULL string.

PRQINFO3 Field - pszComment

pszComment (PSZ)

Queue description.

A NULL string results in no comment. The maximum length is 48 characters (including one byte for the null terminator).

PRQINFO3 Field - fsStatus

fsStatus ([USHORT](#))

Queue status.

PRQ3_PAUSED

Queue is paused (held).

PRQ3_PENDING

Queue is pending deletion.

PRQINFO3 Field - cJobs

cJobs ([USHORT](#))

Number of jobs in queue.

PRQINFO3 Field - pszPrinters

pszPrinters ([PSZ](#))

Print devices connected to queue.

This cannot be NULL.

PRQINFO3 Field - pszDriverName

pszDriverName ([PSZ](#))

Default device driver.

PRQINFO3 Field - pDriverData

pDriverData ([PDRIVDATA](#))

Default queue job properties.

Note: An application can use *pszDriverName*, *pDriverData*, *pszPrProc*, and *pszParms* to construct a valid [DevOpenDC](#) call based only on the queue name.

PRQINFO6

Print-queue information structure.

This structure is used at information level 6.

```
typedef struct _PRQINFO6 {
    PSZ      pszName;          /* Queue name. */
    USHORT   uPriority;         /* Queue priority. */
    USHORT   uStartTime;       /* Minutes after midnight when queue becomes active. */
    USHORT   uUntilTime;       /* Minutes after midnight. when queue ceases to be active. */
    USHORT   fsType;           /* Queue type. */
    PSZ      pszSepFile;        /* Separator-page file. */
    PSZ      pszPrProc;         /* Default queue-processor. */
    PSZ      pszParms;          /* Queue parameters. */
    PSZ      pszComment;        /* Queue description. */
    USHORT   fsStatus;          /* Queue status. */
    USHORT   cJobs;             /* Number of jobs in queue. */
    PSZ      pszPrinters;       /* Print devices connected to queue. */
    PSZ      pszDriverName;     /* Default device driver. */
    PDRIVDATA pDriverData;      /* Default queue job properties. */
    PSZ      pszRemoteComputerName; /* Remote computer name. */
    PSZ      pszRemoteQueueName; /* Remote queue name. */
} PRQINFO6;

typedef PRQINFO6 *PPRQINFO6;
```

PRQINFO6 Field - pszName

pszName (PSZ)

Queue name.

The maximum length of the name in the network case is 256 (including one byte for zero termination).

PRQINFO6 Field - uPriority

uPriority (USHORT)

Queue priority.

The range is 1 through 9, with 1 being the highest queue priority.

The default job priority (DefJobPrio) is determined from:
DefJobPrio=100-(10* *uPriority*).

If a job is added with **PRJ_NO_PRIORITY** specified, DefJobPrio is used. If a default priority higher than the default job priority is specified, the default job priority is used. If a default priority lower than the default is specified, the specified job priority is used.

PRQ_DEF_PRIORITY

Default priority

PRQ_MAX_PRIORITY
Highest priority
PRQ_MIN_PRIORITY
Minimum priority
PRQ_NO_PRIORITY
No priority.

PRQINFO6 Field - uStartTime

uStartTime (USHORT)

Minutes after midnight when queue becomes active.

For example, the value 75 represents 1:15 a.m.

If *uStartTime* and *uUntilTime* are both 0, the print queue is always available.

PRQINFO6 Field - uUntilTime

uUntilTime (USHORT)

Minutes after midnight. when queue ceases to be active.

For example, the value 1200 represents 8 p.m.

If *uUntilTime* and *uStartTime* are both 0, the print queue is always available.

PRQINFO6 Field - fsType

fsType (USHORT)

Queue type.

PRQ3_TYPE_RAW

Data is always enqueued in the device specific format.

PRQ3_TYPE_BYPASS

Allows the spooler to bypass the queue processor and send data directly to the Printer Driver. Setting this bit allows the spooler to print jobs of type PM_Q_RAW while they are still being spooled.

PRQ3_TYPE_APPDEFAULT

This bit is set for the application default queue only.

PRQINFO6 Field - pszSepFile

pszSepFile (PSZ)

Separator-page file.

The path and file name of a separator-page file on the target computer.

This file contains formatting information for the page or pages to be used between print jobs. A relative path name is taken as relative to the current spool directory. A NULL string indicates no separator page.

PRQINFO6 Field - pszPrProc

pszPrProc (PSZ)
Default queue-processor.

PRQINFO6 Field - pszParms

pszParms (PSZ)
Queue parameters.

This can be any text string or a NULL string.

PRQINFO6 Field - pszComment

pszComment (PSZ)
Queue description.

A NULL string results in no comment. The maximum length is 48 characters (including one byte for the null terminator).

PRQINFO6 Field - fsStatus

fsStatus (USHORT)
Queue status.

PRQ3_PAUSED	Queue is paused (held).
PRQ3_PENDING	Queue is pending deletion.

PRQINFO6 Field - cJobs

cJobs (USHORT)

Number of jobs in queue.

PRQINFO6 Field - pszPrinters

pszPrinters ([PSZ](#))

Print devices connected to queue.

This cannot be NULL.

PRQINFO6 Field - pszDriverName

pszDriverName ([PSZ](#))

Default device driver.

PRQINFO6 Field - pDriverData

pDriverData ([PDRIVDATA](#))

Default queue job properties.

Note: An application can use *pszDriverName*, *pDriverData*, *pszPrProc*, and *pszParms* to construct a valid [DevOpenDC](#) call based only on the queue name.

PRQINFO6 Field - pszRemoteComputerName

pszRemoteComputerName ([PSZ](#))

Remote computer name.

The computer name part of a remote queue for which this queue is a local alias.

PRQINFO6 Field - pszRemoteQueueName

pszRemoteQueueName ([PSZ](#))

Remote queue name.

The queue name part of a remote queue for which this queue is a local alias.

PRQPROCINFO

Queue processor information structure (level 0).

```
typedef struct _PRQPROCINFO {  
    CHAR      szQProcName[QNLEN+1]; /* Name of queue processor. */  
} PRQPROCINFO;  
  
typedef PRQPROCINFO *PPRQPROCINFO;
```

PRQPROCINFO Field - szQProcName[QNLEN+1]

szQProcName[QNLEN+1] ([CHAR](#))

Name of queue processor.

This is the name of the queue processor (driver). For example "PMPRINT".

PSZ

Pointer to a null-terminated string.

If you are using C++ **, you may need to use PCSZ.

```
typedef unsigned char *PSZ;
```

PVOID

Pointer to a data type of undefined format.

```
typedef VOID *PVOID;
```

PWPOINT

Pointer to a [WPOINT](#) data structure.

```
#define PWPOINT PPOINTL
```

QMJOBINFO

Information about a job being spooled, returned by [SplQmGetJobID](#).

```
typedef struct _QMJOBINFO {  
    ULONG      ulJobID;          /* Spooler print job identifier. */  
    PSZ        pszComputerName; /* Name of server job being spooled. */  
    PSZ        pszQueueName;    /* Name of queue job being spooled. */  
} QMJOBINFO;  
  
typedef QMJOBINFO *PQMJOBINFO;
```

QMJOBINFO Field - ulJobID

ulJobID ([ULONG](#))

Spooler print job identifier.

This ID may be used with [SplSetJob](#).

QMJOBINFO Field - pszComputerName

pszComputerName ([PSZ](#))

Name of server job being spooled.

This parameter is NULL if the job is spooled locally.

QMJOBINFO Field - pszQueueName

pszQueueName ([PSZ](#))

Name of queue job being spooled.

QMSG

Message structure.

```
typedef struct _QMSG {  
    HWND      hwnd;          /* Window handle. */
```



```
ULONG      msg;          /* Message identity. */
MPARAM     mp1;          /* Parameter 1. */
MPARAM     mp2;          /* Parameter 2. */
ULONG      time;         /* Message time. */
POINTL     ptl;          /* Pointer position when message was generated. */
ULONG      reserved;     /* Reserved. */
} QMSG;

typedef QMSG *PQMSG;
```

QMSG Field - hwnd

hwnd ([HWND](#))
Window handle.

QMSG Field - msg

msg ([ULONG](#))
Message identity.

QMSG Field - mp1

mp1 ([MPARAM](#))
Parameter 1.

QMSG Field - mp2

mp2 ([MPARAM](#))
Parameter 2.

QMSG Field - time

time ([ULONG](#))
Message time.

QMSG Field - ptl

ptl ([POINTL](#))

Pointer position when message was generated.

QMSG Field - reserved

reserved ([ULONG](#))

Reserved.

QUERYRECFROMRECT

Structure that contains information about a container record that is bounded by a specified rectangle. This structure is used in the [CM_QUERYRECORDFROMRECT](#) container message only.

```
typedef struct _QUERYRECFROMRECT {  
    ULONG    cb;           /* Structure size. */  
    RECTL    rect;         /* Rectangle. */  
    ULONG    fsSearch;     /* Search control flags. */  
} QUERYRECFROMRECT;
```

```
typedef QUERYRECFROMRECT *PQUERYRECFROMRECT;
```

QUERYRECFROMRECT Field - cb

cb ([ULONG](#))

Structure size.

The size (in bytes) of the QUERYRECFROMRECT data structure.

QUERYRECFROMRECT Field - rect

rect ([RECTL](#))

Rectangle.

The rectangle to query, in virtual coordinates relative to the container window origin. If the details view (CV_DETAIL) is displayed, the x-coordinates of the rectangle are ignored.

QUERYRECFROMRECT Field - fsSearch

fsSearch (ULONG)

Search control flags.

One flag from each of the following groups can be specified:

- Search sensitivity:
 - CMA_COMPLETE Returns the container records that are completely within the bounding rectangle.
 - CMA_PARTIAL Returns the container records that are completely or partially within the bounding rectangle.
- Enumeration order:
 - CMA_ITEMORDER Container records are enumerated in item order, lowest to highest.
 - CMA_ZORDER Container records are enumerated by z-order, from top to bottom. This flag is valid for the icon view only.

QUERYRECORDRECT

Structure that contains information about the rectangle of the specified container record, relative to the container window origin. This structure is used in the [CM_QUERYRECORDRECT](#) container message only.

```
typedef struct _QUERYRECORDRECT {
    ULONG      cb;           /* Structure size. */
    PRECORDCORE pRecord;     /* Pointer. */
    ULONG      fRightSplitWindow; /* Window flag. */
    ULONG      fsExtent;     /* Rectangle flags. */
} QUERYRECORDRECT;

typedef QUERYRECORDRECT *PQUERYRECORDRECT;
```

QUERYRECORDRECT Field - cb

cb (ULONG)

Structure size.

The size (in bytes) of the QUERYRECORDRECT structure.

QUERYRECORDRECT Field - pRecord

pRecord ([PRECORDCORE](#))
Pointer.

Pointer to the specified [RECORDCORE](#) data structure.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

QUERYRECORDRECT Field - fRightSplitWindow

fRightSplitWindow ([ULONG](#))
Window flag.

Flag that specifies the right or left window in the split details view.

This flag is ignored if the view is not the split details view.

- | | |
|-------|---------------------------------|
| TRUE | Right split window is returned. |
| FALSE | Left split window is returned. |

QUERYRECORDRECT Field - fsExtent

fsExtent ([ULONG](#))
Rectangle flags.

Flags that specify the extent of the desired rectangle.

These flags can be combined by using a logical OR operator (|) to return the rectangle that bounds the icon, the expanded and collapsed icon or bit map, and the text.

- | | |
|--------------|---|
| CMA_ICON | Returns the icon rectangle. |
| CMA_TEXT | Returns the text rectangle. |
| CMA_TREEICON | Returns the rectangle of the expanded and collapsed icons or bit maps. This flag is valid for the tree icon and tree text views only. |

RECORDCORE

Structure that contains information for records in a container control. This data structure is used if the CCS_MINIRECORDCORE style bit is not specified when a container is created.

```
typedef struct _RECORDCORE {
    ULONG cb; /* The size, in bytes, of the RECORDCORE structure. */
    ULONG flRecordAttr; /* Container record attributes. */
    POINTL ptlIcon; /* Position of a container record in the icon view. */
    struct _RECORDCORE *preccNextRecord; /* Pointer to the next linked record. */
    PSZ pszIcon; /* Text for the icon view (CV_ICON). */
}
```

```

HPOINTER          hptrIcon;          /* Icon that is displayed when the CV_MINI style bit is not speci
HPOINTER          hptrMiniIcon;      /* Icon that is displayed when the CV_MINI style bit is specified
HBITMAP           hbmBitmap;         /* Bit map displayed when the CV_MINI style bit is not specified.
HBITMAP           hbmMiniBitmap;     /* Bit map displayed when the CV_MINI style bit is specified. */
PTREEITEMDESC     pTreeItemDesc;     /* Pointer to a TREEITEMDESC structure. */
PSZ               pszText;           /* Text for the text view (CV_TEXT). */
PSZ               pszName;           /* Text for the name view (CV_NAME). */
PSZ               pszTree;           /* Text for the tree view (CV_TREE). */
} RECORDCORE;

typedef RECORDCORE *PRECORDCORE;

```

RECORDCORE Field - cb

cb ([ULONG](#))
The size, in bytes, of the RECORDCORE structure.

RECORDCORE Field - flRecordAttr

flRecordAttr ([ULONG](#))
Container record attributes.

This parameter can contain any or all of the following:

CRA_COLLAPSED
Specifies that a record is collapsed.

CRA_CURSORED
Specifies that a record will be drawn with a selection cursor.

CRA_DISABLED
Specifies that a record will be drawn with unavailable-state emphasis.

CRA_DROPONABLE
Specifies that a record can be a target for direct manipulation.

CRA_EXPANDED
Specifies that a record is expanded.

CRA_FILTERED
Specifies that a record is filtered and, therefore, hidden from view.

CRA_INUSE
Specifies that a record will be drawn with in-use emphasis.

CRA_PICKED
Specifies that the container record willl be picked up as part of the drag set.

CRA_SELECTED
Specifies that a record will be drawn with selected-state emphasis.

CRA_SOURCE
Specifies that a record will be drawn with source-menu emphasis.

RECORDCORE Field - pttlIcon

ptlIcon ([POINTL](#))
Position of a container record in the icon view.

RECORDCORE Field - preccNextRecord

preccNextRecord (struct _RECORDCORE *)
Pointer to the next linked record.

RECORDCORE Field - pszIcon

pszIcon ([PSZ](#))
Text for the icon view (CV_ICON).

RECORDCORE Field - hptrIcon

hptrIcon ([HPOINTER](#))
Icon that is displayed when the CV_MINI style bit is not specified.

This field is used when the CA_DRAWICON container attribute of the [CNRINFO](#) data structure is set.

RECORDCORE Field - hptrMinilcon

hptrMinilcon ([HPOINTER](#))
Icon that is displayed when the CV_MINI style bit is specified.

This field is used when the CA_DRAWICON container attribute of the [CNRINFO](#) data structure is set.

RECORDCORE Field - hbmBitmap

hbmBitmap ([HBITMAP](#))
Bit map displayed when the CV_MINI style bit is not specified.

This field is used when the CA_DRAWBITMAP container attribute of the [CNRINFO](#) data structure is set.

RECORDCORE Field - hbmMiniBitmap

hbmMiniBitmap ([HBITMAP](#))
Bit map displayed when the CV_MINI style bit is specified.

This field is used when the CA_DRAWBITMAP container attribute of the [CNRINFO](#) data structure is set.

RECORDCORE Field - pTreeltemDesc

pTreeltemDesc ([PTREEITEMDESC](#))
Pointer to a [TREEITEMDESC](#) structure.

The [TREEITEMDESC](#) structure contains the icons and bit maps used to represent the state of an expanded or collapsed parent item in the tree name view.

RECORDCORE Field - pszText

pszText ([PSZ](#))
Text for the text view (CV_TEXT).

RECORDCORE Field - pszName

pszName ([PSZ](#))
Text for the name view (CV_NAME).

RECORDCORE Field - pszTree

pszTree ([PSZ](#))
Text for the tree view (CV_TREE).

RECORDINSERT

Structure that contains information about [RECORDCORE](#) structures that are being inserted into a container. The RECORDINSERT structure is used in the [CM_INSERTRECORD](#) container message only.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then [MINIRECORDCORE](#) should be used instead of [RECORDCORE](#) and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

```
typedef struct _RECORDINSERT {
    ULONG      cb; /* Structure size. */
    PRECORDCORE pRecordOrder; /* Record order. */
    PRECORDCORE pRecordParent; /* Pointer to record parent. */
    ULONG      fInvalidateRecord; /* Update flag. */
    ULONG      zOrder; /* Record z-order. */
    ULONG      cRecordsInsert; /* Number of root-level structures. */
} RECORDINSERT;

typedef RECORDINSERT *PRECORDINSERT;
```

RECORDINSERT Field - cb

cb ([ULONG](#))

Structure size.

The size (in bytes) of the RECORDINSERT structure.

RECORDINSERT Field - pRecordOrder

pRecordOrder ([PRECORDCORE](#))

Record order.

Orders the [RECORDCORE](#) structures relative to other [RECORDCORE](#) structures in the container. The values can be:

CMA_FIRST

Places a [RECORDCORE](#) structure, or list of [RECORDCORE](#) structures, at the beginning of the list of structures.

CMA_END

Places a [RECORDCORE](#) structure, or list of [RECORDCORE](#) structures, at the end of the list of structures.

Other

Pointer to a [RECORDCORE](#) structure after which this structure, or list of structures, is to be inserted.

RECORDINSERT Field - pRecordParent

pRecordParent ([PRECORDCORE](#))

Pointer to record parent.

Pointer to a [RECORDCORE](#) structure that is the parent of the records to be inserted. This field is used only with the CMA_FIRST or

CMA_END attributes of the *pRecordOrder* field.

RECORDINSERT Field - flInvalidateRecord

flInvalidateRecord (ULONG)

Update flag.

Flag that indicates an automatic display update after RECORDCORE structures are inserted.

TRUE

The display is automatically updated after a RECORDCORE structure is inserted.

FALSE

The application must send the CM_INVALIDATERECORD message after a RECORDCORE structure is inserted.

RECORDINSERT Field - zOrder

zOrder (ULONG)

Record z-order.

Positions the RECORDCORE structure in z-order, relative to other records in the container. The values can be:

CMA_TOP

Places a RECORDCORE structure at the top of the z-order. This is the default value.

CMA_BOTTOM

Places a RECORDCORE structure at the bottom of the z-order.

RECORDINSERT Field - cRecordsInsert

cRecordsInsert (ULONG)

Number of root-level structures.

The number of root-level RECORDCORE structures to be inserted. The *cRecordsInsert* field value must be greater than 0.

RECTL

Rectangle structure.

```
typedef struct _RECTL {  
    LONG    xLeft;    /* X-coordinate of left-hand edge of rectangle. */  
    LONG    yBottom;  /* Y-coordinate of bottom edge of rectangle. */  
    LONG    xRight;   /* X-coordinate of right-hand edge of rectangle. */  
    LONG    yTop;     /* Y-coordinate of top edge of rectangle. */  
} RECTL;
```

```
typedef RECTL *PRECTL;
```

RECTL Field - xLeft

xLeft ([LONG](#))
X-coordinate of left-hand edge of rectangle.

RECTL Field - yBottom

yBottom ([LONG](#))
Y-coordinate of bottom edge of rectangle.

RECTL Field - xRight

xRight ([LONG](#))
X-coordinate of right-hand edge of rectangle.

RECTL Field - yTop

yTop ([LONG](#))
Y-coordinate of top edge of rectangle.

RENDERFILE

File-rendering structure.

```
typedef struct _RENDERFILE {  
    HWND      hwndDragFiles; /* Conversation handle. */  
    HSTR      hstrSource;    /* Handle to source file name. */  
    HSTR      hstrTarget;    /* Handle to target file name. */  
    USHORT    fMove;         /* Operation. */  
    USHORT    usRsvd;        /* Reserved. */  
} RENDERFILE;  
  
typedef RENDERFILE *PRENDERFILE;
```

RENDERFILE Field - hwndDragFiles

hwndDragFiles ([HWND](#))
Conversation handle.

Created by [DrgDragFiles](#).

RENDERFILE Field - hstrSource

hstrSource ([HSTR](#))
Handle to source file name.

RENDERFILE Field - hstrTarget

hstrTarget ([HSTR](#))
Handle to target file name.

RENDERFILE Field - fMove

fMove ([USHORT](#))
Operation.

TRUE	Move the file.
FALSE	Copy the file.

RENDERFILE Field - usRsvd

usRsvd ([USHORT](#))
Reserved.

RGB

RGB color value.

```
typedef struct _RGB {  
    BYTE    bBlue; /* Blue component of the color definition. */  
    BYTE    bGreen; /* Green component of the color definition. */  
    BYTE    bRed; /* Red component of the color definition. */  
} RGB;  
  
typedef RGB *PRGB;
```

RGB Field - bBlue

bBlue (BYTE)
Blue component of the color definition.

RGB Field - bGreen

bGreen (BYTE)
Green component of the color definition.

RGB Field - bRed

bRed (BYTE)
Red component of the color definition.

RGB2

RGB color value.

```
typedef struct _RGB2 {  
    BYTE    bBlue; /* Blue component of the color definition. */  
    BYTE    bGreen; /* Green component of the color definition. */  
    BYTE    bRed; /* Red component of the color definition. */  
    BYTE    fcOptions; /* Entry options. */  
} RGB2;  
  
typedef RGB2 *PRGB2;
```

RGB2 Field - bBlue

bBlue (BYTE)
Blue component of the color definition.

RGB2 Field - bGreen

bGreen (BYTE)
Green component of the color definition.

RGB2 Field - bRed

bRed (BYTE)
Red component of the color definition.

RGB2 Field - fcOptions

fcOptions (BYTE)
Entry options.

These can be ORed together if required:

PC_RESERVED

The color entry is reserved for animating color with the palette manager.

PC_EXPLICIT

The low-order word of the color table entry designates a physical palette slot. This allows an application to show the actual contents of the device palette as realized for other logical palettes. This does not prevent the color in the slot from being changed for any reason.

RGNRECT

Region-rectangle structure.

```
typedef struct _RGNRECT {
    ULONG    ircStart;    /* Rectangle number from which to start enumerating. */
    ULONG    crc;         /* Number of rectangles that can be returned. */
    ULONG    crcReturned; /* Number of rectangles returned. */
    ULONG    ulDirection; /* Direction in which the returned rectangles are to be ordered. */
} RGNRECT;
```

```
typedef RGNRECT *PRGNRECT;
```

RGNRECT Field - ircStart

ircStart ([ULONG](#))
Rectangle number from which to start enumerating.
Numbering starts from 1.

RGNRECT Field - crc

crc ([ULONG](#))
Number of rectangles that can be returned.
This must be 1 or greater.

RGNRECT Field - crcReturned

crcReturned ([ULONG](#))
Number of rectangles returned.
A value of less than *crc* indicates that there are no more rectangles to enumerate.

RGNRECT Field - ulDirection

ulDirection ([ULONG](#))
Direction in which the returned rectangles are to be ordered.
This ordering uses the leading edge of a rectangle:

RECTDIR_LFRT_TOPBOT	Left-to-right, top-to-bottom
RECTDIR_RTLF_TOPBOT	Right-to-left, top-to-bottom
RECTDIR_LFRT_BOTTOM	Left-to-right, bottom-to-top
RECTDIR_RTLF_BOTTOM	Right-to-left, bottom-to-top

SBCDATA

Scroll-bar control data structure.

```
typedef struct _SBCDATA {
    USHORT    cb;           /* Length of control data in bytes. */
    USHORT    sHilite;      /* Highlighting code. */
    SHORT     posFirst;     /* First bound of the scroll-bar range. */
    SHORT     posLast;      /* Last bound of the scroll-bar range. */
    SHORT     posThumb;      /* Slider position. */
    SHORT     cVisible;     /* Number of data items visible. */
    SHORT     cTotal;       /* Number of data items available. */
} SBCDATA;

typedef SBCDATA *PSBCDATA;
```

SBCDATA Field - cb

cb (USHORT)

Length of control data in bytes.

The length of the control data for a scroll-bar control.

This indicates which part of the scroll bar is to be highlighted, if any.

SBCDATA Field - sHilite

sHilite (USHORT)

Highlighting code.

ZERO	No highlighting
SB_LINEUP	Line up arrow
SB_LINELEFT	Line left arrow
SB_LINEDOWN	Line down arrow
SB_LINERIGHT	Line right arrow
SB_PAGEUP	Page up arrow
SB_PAGELEFT	Page left arrow
SB_PAGEDOWN	Page down arrow
SB_PAGERIGHT	Page right arrow
SB_SLIDERTRACK	Slider.

SBCDATA Field - posFirst

posFirst ([SHORT](#))

First bound of the scroll-bar range.

SBCDATA Field - posLast

posLast ([SHORT](#))

Last bound of the scroll-bar range.

SBCDATA Field - posThumb

posThumb ([SHORT](#))

Slider position.

SBCDATA Field - cVisible

cVisible ([SHORT](#))

Number of data items visible.

SBCDATA Field - cTotal

cTotal ([SHORT](#))

Number of data items available.

SEARCHSTRING

Structure that contains information about the container text string that is the object of the search. This structure is used in the [CM_SEARCHSTRING](#) container message only.

```
typedef struct _SEARCHSTRING {  
    ULONG    cb;                /* Structure size. */  
    PSZ      pszSearch;         /* Pointer to the search string. */  
    ULONG    fsPrefix;          /* Search flag. */  
    ULONG    fsCaseSensitive;    /* Case sensitivity flag. */  
    ULONG    usView;            /* View to search. */  
} SEARCHSTRING;
```



```
typedef SEARCHSTRING *PSEARCHSTRING;
```

SEARCHSTRING Field - cb

cb (ULONG)
Structure size.
The size (in bytes) of the SEARCHSTRING structure.

SEARCHSTRING Field - pszSearch

pszSearch (PSZ)
Pointer to the search string.

SEARCHSTRING Field - fsPrefix

fsPrefix (ULONG)
Search flag.
Search flag that defines the criteria by which the string specified by the *pszSearch* field is to be compared with the text of the container records to determine the pointer to the first matching record.

TRUE	Matching occurs if the leading characters of the container record are the characters specified by the <i>pszSearch</i> field.
FALSE	Matching occurs if the container record contains a substring of the characters specified by the <i>pszSearch</i> field.

SEARCHSTRING Field - fsCaseSensitive

fsCaseSensitive (ULONG)
Case sensitivity flag.
Determines case sensitivity of the search.

TRUE	The search is case sensitive.
FALSE	The search is not case sensitive.

SEARCHSTRING Field - usView

usView (ULONG)

View to search.

Search one of the container views for the string. Valid values are:

- CV_ICON
- CV_NAME
- CV_TEXT
- CV_TREE
- CV_DETAIL.

SEGOFF

Two-byte segment offset in bytes.

```
typedef follow SEGOFF;
```

SHORT

Signed integer in the range -32 768 through 32 767.

```
#define SHORT short
```

SIZEF

Size structure (FIXED values).

```
typedef struct _SIZEF {  
    FIXED    cx; /* Width. */  
    FIXED    cy; /* Height. */  
} SIZEF;
```

```
typedef SIZEF *PSIZEF;
```

SIZEF Field - cx

cx (FIXED)

Width.

SIZEF Field - cy

cy (FIXED)
Height.

SIZEL

Size structure (LONG values).

```
typedef struct _SIZEL {
    LONG      cx; /* Width. */
    LONG      cy; /* Height. */
} SIZEL;

typedef SIZEL *PSIZEL;
```

SIZEL Field - cx

cx (LONG)
Width.

SIZEL Field - cy

cy (LONG)
Height.

SLDCDATA

Slider control data structure.

```
typedef struct _SLDCDATA {
    ULONG      cbSize; /* Data length. */
    USHORT     usScale1Increments; /* Scale increments. */
    USHORT     usScale1Spacing; /* Scale spacing. */
    USHORT     usScale2Increments; /* Alternate scale increments. */
    USHORT     usScale2Spacing; /* Alternate scale spacing. */
} SLDCDATA;
```

typedef SLDCDATA *PSLDCDATA;

SLDCDATA Field - cbSize

cbSize (ULONG)
Data length.

Length of the control data in bytes.

SLDCDATA Field - usScale1Increments

usScale1Increments (USHORT)
Scale increments.

The number of increments to set for the slider control. This number represents the range of values that can be selected within the slider when the SLS_PRIMARYSCALE1 style bit is specified.

SLDCDATA Field - usScale1Spacing

usScale1Spacing (USHORT)
Scale spacing.

The spacing between increments, expressed in pixels. It represents the unit that is the smallest division of the scale when the SLS_PRIMARYSCALE1 style bit is specified. If 0 is specified, the slider automatically calculates the spacing based on the window size and the number of increments specified.

SLDCDATA Field - usScale2Increments

usScale2Increments (USHORT)
Alternate scale increments.

An alternate number of increments to set for the slider control. This number represents the range of values that can be selected within the slider when the SLS_PRIMARYSCALE2 style bit is specified.

SLDCDATA Field - usScale2Spacing

usScale2Spacing ([USHORT](#))
Alternate scale spacing.

An alternate spacing between increments, expressed in pixels. It represents the unit that is the smallest division of the scale when the SLS_PRIMARYSCALE2 style bit is specified. If 0 is specified, the slider automatically calculates the spacing based on the window size and the number of increments specified.

SMHSTRUCT

Send-message-hook structure.

```
typedef struct _SMHSTRUCT {
    MPARAM    mp2;    /* Parameter 2. */
    MPARAM    mp1;    /* Parameter 1. */
    ULONG     msg;     /* Message identity. */
    HWND      hwnd;    /* Window handle. */
    ULONG     model;   /* Message identity. */
} SMHSTRUCT;

typedef SMHSTRUCT *PSMHSTRUCT;
```

SMHSTRUCT Field - mp2

mp2 ([MPARAM](#))
Parameter 2.

SMHSTRUCT Field - mp1

mp1 ([MPARAM](#))
Parameter 1.

SMHSTRUCT Field - msg

msg ([ULONG](#))
Message identity.

SMHSTRUCT Field - hwnd

hwnd ([HWND](#))
Window handle.

SMHSTRUCT Field - model

model ([ULONG](#))
Message identity.

SPBCDATA

Spin button control data structure.

```
typedef struct _SPBCDATA {  
    ULONG    cbSize; /* Size of control block. */  
    ULONG    ulTextLimit; /* Entryfield text limit. */  
    LONG     lLowerLimit; /* Spin lower limit (numeric only). */  
    LONG     lUpperLimit; /* Spin upper limit (numeric only). */  
    ULONG    idMasterSpb; /* ID of the servant's master spinbutton. */  
    PVOID    pHWXctlData; /* Handwriting control data structure flag. */  
} SPBCDATA;  
  
typedef SPBCDATA *PSPBCDATA;
```

The SPBCDATA structure is used in the **ctldata** parameter of [WinCreateWindow](#).

When using this structure the [SPBM_SETLIMITS](#), [SPBM_SETTEXTLIMIT](#), and [SPBM_SETMASTER](#) messages do not need to be specified.

- *ulTextLimit* and *lLowerLimit* replace [SPBM_SETLIMITS](#).
 - *lUpperLimit* replaces [SPBM_SETTEXTLIMIT](#).
 - *idMasterSpb* replaces [SPBM_SETMASTER](#).
-

SPBCDATA Field - cbSize

cbSize ([ULONG](#))
Size of control block.

SPBCDATA Field - ulTextLimit

ulTextLimit ([ULONG](#))
Entryfield text limit.

SPBCDATA Field - ILowerLimit

ILowerLimit (LONG)
Spin lower limit (numeric only).

SPBCDATA Field - IUpperLimit

IUpperLimit (LONG)
Spin upper limit (numeric only).

SPBCDATA Field - idMasterSpb

idMasterSpb (ULONG)
ID of the servant's master spinbutton.

SPBCDATA Field - pHWXCtlData

pHWXCtlData (PVOID)
Handwriting control data structure flag.

Flag that indicates whether this structure is followed by a handwriting control data structure or a pointer to a handwriting control data structure. Possible values are:

PENPMDATAF	Handwriting control data structure.
PENPMDATA	Pointer to the handwriting control data structure.

SPLERR

Error value in the range 0 to 65 535.

```
typedef unsigned long SPLERR;
```

STR8

String of eight characters.

```
typedef CHAR STR8[8];
```

STR16

String of characters, with an implicit length, in a 16-byte field.

```
typedef CHAR STR16[16];
```

STR32

String of characters, with an implicit length, in a 32-byte field.

```
typedef CHAR STR32[32];
```

STR64

String of characters, with an implicit length, in a 64-byte field.

```
typedef CHAR STR64[64];
```

STYLECHANGE

Style-change structure. This structure is returned by the [FNTM_STYLECHANGED](#) message.

All "old" fields describe the style attributes before the user made a change. The other, or "new", parameters describe the style that will be in effect after this is passed to [WinDefFontDlgProc](#). When the "old" and "new" values are the same, the user made no change.

For further details of the parameters, see [FONTDLG](#).

```
typedef struct _STYLECHANGE {
    USHORT    usWeight;           /* New weight of font. */
    USHORT    usWeightOld;        /* Old weight of font. */
    USHORT    usWidth;           /* New width of font. */
    USHORT    usWidthOld;        /* Old width of font. */
    ULONG     flType;            /* New type of font. */
    ULONG     flTypeOld;         /* Old type of font. */
    ULONG     flTypeMask;        /* New type mask. */
    ULONG     flTypeMaskOld;     /* Old type mask. */
    ULONG     flStyle;           /* New selected style bits. */
    ULONG     flStyleOld;        /* Old selected style bits. */
    ULONG     flStyleMask;       /* New mask of style bits to use. */
}
```



```
    ULONG         flStyleMaskOld; /* Old mask of style bits to use. */
} STYLECHANGE;

typedef STYLECHANGE *PSTYLECHANGE;
```

STYLECHANGE Field - usWeight

usWeight (USHORT)
New weight of font.

STYLECHANGE Field - usWeightOld

usWeightOld (USHORT)
Old weight of font.

STYLECHANGE Field - usWidth

usWidth (USHORT)
New width of font.

STYLECHANGE Field - usWidthOld

usWidthOld (USHORT)
Old width of font.

STYLECHANGE Field - flType

flType (ULONG)
New type of font.

STYLECHANGE Field - flTypeOld

flTypeOld (ULONG)
Old type of font.

STYLECHANGE Field - flTypeMask

flTypeMask (ULONG)
New type mask.

STYLECHANGE Field - flTypeMaskOld

flTypeMaskOld (ULONG)
Old type mask.

STYLECHANGE Field - flStyle

flStyle (ULONG)
New selected style bits.

STYLECHANGE Field - flStyleOld

flStyleOld (ULONG)
Old selected style bits.

STYLECHANGE Field - flStyleMask

flStyleMask (ULONG)
New mask of style bits to use.

STYLECHANGE Field - flStyleMaskOld

flStyleMaskOld ([ULONG](#))
Old mask of style bits to use.

SWBLOCK

Switch-list block structure.

```
typedef struct _SWBLOCK {
    ULONG      cswentry; /* Count of switch list entries. */
    SWENTRY    aswentry[1]; /* Switch list entries. */
} SWBLOCK;

typedef SWBLOCK *PSWBLOCK;
```

SWBLOCK Field - cswentry

cswentry ([ULONG](#))
Count of switch list entries.

SWBLOCK Field - aswentry[1]

aswentry[1] ([SWENTRY](#))
Switch list entries.

SWCNTRL

Switch-list control block structure.

```
typedef struct _SWCNTRL {
    HWND      hwnd; /* Window handle. */
    HWND      hwndIcon; /* Window-handle icon. */
    HPROGRAM   hprog; /* Program handle. */
    PID       idProcess; /* Process identity. */
    ULONG     idSession; /* Session identity. */
    ULONG     uchVisibility; /* Visibility. */
    ULONG     fbJump; /* Jump indicator. */
    CHAR      szSwttitle[MAXNAMEL+4]; /* Switch-list control block title (null-terminated). */
    ULONG     bProgType; /* Program type. */
} SWCNTRL;

typedef SWCNTRL *PSWCNTRL;
```

SWCCTRL Field - hwnd

hwnd (HWND)
Window handle.

SWCCTRL Field - hwndIcon

hwndIcon (HWND)
Window-handle icon.

SWCCTRL Field - hprog

hprog (HPROGRAM)
Program handle.

SWCCTRL Field - idProcess

idProcess (PID)
Process identity.

SWCCTRL Field - idSession

idSession (ULONG)
Session identity.

SWCCTRL Field - uchVisibility

uchVisibility (ULONG)
Visibility:

SWL_VISIBLEVisible in startup list

SWL_INVISIBLEInvisible in startup list

SWL_GRAYEDItem cannot be switched to (note that it is not actually grayed in the list).

SWCNTRL Field - fbJump

fbJump (ULONG)
Jump indicator:

SWL_JUMPABLEParticipates in jump sequence

SWL_NOTJUMPABLEDoes not participate in jump sequence.

SWCNTRL Field - szSwtitle[MAXNAMEL+4]

szSwtitle[MAXNAMEL+4] (CHAR)
Switch-list control block title (null-terminated).

SWCNTRL Field - bProgType

bProgType (ULONG)
Program type.

Possible values are:

PROG_DEFAULT0

PROG_FULLSCREEN1

PROG_WINDOWABLEVIO2

PROG_PM3

PROG_VDM4

PROG_WINDOWEDVDM7

Although there are several other program types for WIN-OS/2 programs, these do not show up in this structure. Instead, the PROG_VDM or PROG_WINDOWEDVDM program types are used. For instance, for PROG_31_STDSEAMLESSVDM, PROG_WINDOWEDVDM is used. This is because all the WIN-OS/2 programs run in DOS sessions. For example, if a program is a windowed WIN-OS/2 program, it runs in a PROG_WINDOWEDVDM session. Likewise, if it's a full-screen WIN-OS/2 program, it runs in a PROG_VDM session.

SWENTRY

Switch-list entry structure.

```
typedef struct _SWENTRY {
    HSWITCH    hswitch; /* Switch-list entry handle. */
    SWCNTRL    swctl;   /* Switch-list control block structure. */
} SWENTRY;

typedef SWENTRY *PSWENTRY;
```

SWENTRY Field - hswitch

hswitch (HSWITCH)
Switch-list entry handle.

SWENTRY Field - swctl

swctl (SWCNTRL)
Switch-list control block structure.

SWP

Set-window-position structure.

```
typedef struct _SWP {
    ULONG    fl; /* Options. */
    LONG     cy; /* Window height. */
    LONG     cx; /* Window width. */
    LONG     y;  /* Y-coordinate of origin. */
    LONG     x;  /* X-coordinate of origin. */
    HWND     hwndInsertBehind; /* Window behind which this window is placed. */
    HWND     hwnd; /* Window handle. */
    ULONG     ulReserved1; /* Reserved value; must be 0. */
    ULONG     ulReserved2; /* Reserved value; must be 0. */
} SWP;

typedef SWP *PSWP;
```

SWP Field - fl

fl (ULONG)
Options.

Possible values are shown in the following list:

SWP_ACTIVATE

SWP_DEACTIVATE
SWP_HIDE
SWP_MAXIMIZE
SWP_MINIMIZE
SWP_MOVE
SWP_NOADJUST
SWP_NOERASEWINDOW
SWP_NOREDRAW
SWP_RESTORE
SWP_SHOW
SWP_SIZE
SWP_ZORDER

SWP Field - cy

cy (LONG)
Window height.

SWP Field - cx

cx (LONG)
Window width.

SWP Field - y

y (LONG)
Y-coordinate of origin.

SWP Field - x

x (LONG)

X-coordinate of origin.

SWP Field - hwndInsertBehind

hwndInsertBehind ([HWND](#))

Window behind which this window is placed.

SWP Field - hwnd

hwnd ([HWND](#))

Window handle.

SWP Field - ulReserved1

ulReserved1 ([ULONG](#))

Reserved value; must be 0.

SWP Field - ulReserved2

ulReserved2 ([ULONG](#))

Reserved value; must be 0.

TID

Thread identity.

```
typedef LHANDLE TID;
```

TRACKINFO

Tracking-information structure.


```
typedef struct _TRACKINFO {
    LONG    cxBorder;        /* Border width. */
    LONG    cyBorder;        /* Border height. */
    LONG    cxGrid;         /* Grid width. */
    LONG    cyGrid;         /* Grid height. */
    LONG    cxKeyboard;      /* Character cell width movement for arrow key. */
    LONG    cyKeyboard;      /* Character cell height movement for arrow key. */
    RECTL   rclTrack;        /* Starting tracking rectangle. */
    RECTL   rclBoundary;     /* Boundary rectangle. */
    POINTL  ptlMinTrackSize; /* Minimum tracking size. */
    POINTL  ptlMaxTrackSize; /* Maximum tracking size. */
    ULONG   fs;             /* Tracking options. */
} TRACKINFO;

typedef TRACKINFO *PTRACKINFO;
```

TRACKINFO Field - cxBorder

cxBorder ([LONG](#))
Border width.

The width of the left and right tracking sides.

TRACKINFO Field - cyBorder

cyBorder ([LONG](#))
Border height.

The height of the top and bottom tracking sides.

TRACKINFO Field - cxGrid

cxGrid ([LONG](#))
Grid width.

The horizontal bounds of the tracking movements.

TRACKINFO Field - cyGrid

cyGrid ([LONG](#))
Grid height.

The vertical bounds of the tracking movements.

TRACKINFO Field - cxKeyboard

cxKeyboard ([LONG](#))

Character cell width movement for arrow key.

TRACKINFO Field - cyKeyboard

cyKeyboard ([LONG](#))

Character cell height movement for arrow key.

TRACKINFO Field - rcTrack

rcTrack ([RECTL](#))

Starting tracking rectangle.

This is modified as the rectangle is tracked and holds the new tracking position, when tracking is complete.

TRACKINFO Field - rcBoundary

rcBoundary ([RECTL](#))

Boundary rectangle.

This is an absolute bounding rectangle that the tracking rectangle cannot extend; see also `TF_ALLINBOUNDARY`.

TRACKINFO Field - ptlMinTrackSize

ptlMinTrackSize ([POINTL](#))

Minimum tracking size.

TRACKINFO Field - ptlMaxTrackSize

ptlMaxTrackSize ([POINTL](#))
Maximum tracking size.

TRACKINFO Field - fs

fs ([ULONG](#))
Tracking options.

In alphabetic order:

- TF_ALLINBOUNDARY**
The default tracking is such that some part of the tracking rectangle is within the bounding rectangle defined by *rcBoundary*. This minimum size is defined by *cxBorder* and *cyBorder*.
- If **TF_ALLINBOUNDARY** is specified, the tracking is performed so that no part of the tracking rectangle ever falls outside of the bounding rectangle.
- TF_BOTTOM**
Track the bottom side of the rectangle.
- TF_GRID**
Tracking is restricted to the grid defined by *cxGrid* and *cyGrid*.
- TF_LEFT**
Track the left side of the rectangle.
- TF_MOVE**
Track all sides of the rectangle.
- TF_RIGHT**
Track the right side of the rectangle.
- TF_SETPINTERPOS**
The pointer is repositioned according to other flags as follows:
- | | |
|------------------|---|
| none | Pointer is centered in the tracking rectangle. |
| TF_MOVE | Pointer is centered in the tracking rectangle. |
| TF_LEFT | Pointer is vertically centered at the left of the tracking rectangle. |
| TF_TOP | Pointer is horizontally centered at the top of the tracking rectangle. |
| TF_RIGHT | Pointer is vertically centered at the right of the tracking rectangle. |
| TF_BOTTOM | Pointer is horizontally centered at the bottom of the tracking rectangle. |
- TF_STANDARD**
cx, *cy*, *cxGrid*, and *cyGrid* are all multiples of *cxBorder* and *cyBorder*.
- TF_TOP**
Track the top side of the rectangle.

TREEITEMDESC

Structure that contains icons and bit maps used to represent the state of an expanded or collapsed parent item in the tree name view of a container control.

```
typedef struct _TREEITEMDESC {
    HBITMAP    hbmExpanded;    /* Expanded bit-map handle. */
    HBITMAP    hbmCollapsed;  /* Collapsed bit-map handle. */
    HPOINTER   hptrExpanded;   /* Expanded icon handle. */
    HPOINTER   hptrCollapsed;  /* Collapsed icon handle. */
} TREEITEMDESC;

typedef TREEITEMDESC *PTREEITEMDESC;
```

TREEITEMDESC Field - hbmExpanded

hbmExpanded ([HBITMAP](#))

Expanded bit-map handle.

The handle of the bit map to be used to represent an expanded parent item in the tree name view.

TREEITEMDESC Field - hbmCollapsed

hbmCollapsed ([HBITMAP](#))

Collapsed bit-map handle.

The handle of the bit map to be used to represent a collapsed parent item in the tree name view.

TREEITEMDESC Field - hptrExpanded

hptrExpanded ([HPOINTER](#))

Expanded icon handle.

The handle of the icon to be used to represent an expanded parent item in the tree name view.

TREEITEMDESC Field - hptrCollapsed

hptrCollapsed ([HPOINTER](#))

Collapsed icon handle.

The handle of the icon to be used to represent a collapsed parent item in the tree name view.

TREEMOVE

Data structure for moving nodes in the tree to a new parent.

```
typedef struct _TREEMOVE {
    PRECORDCORE    preccMove;      /* Record to be moved. */
    PRECORDCORE    preccNewParent; /* New parent for preccMove. */
    PRECORDCORE    pRecordOrder;   /* Record order for siblings. */
    BOOL           flMoveSiblings; /* Flag indicating whether to move siblings. */
} TREEMOVE;

typedef TREEMOVE *PTREEMOVE;
```

TREEMOVE Field - preccMove

preccMove (PRECORDCORE)
Record to be moved.

TREEMOVE Field - preccNewParent

preccNewParent (PRECORDCORE)
New parent for *preccMove*.

TREEMOVE Field - pRecordOrder

pRecordOrder (PRECORDCORE)
Record order for siblings.

Possible values are described in the following list:

- CMA_FIRST *preccMove* moves to the FIRST child position of *preccNewParent*. If *preccNewParent* is NULL, *preccMove* becomes the first root level record of the container.
- CMA_LAST *preccMove* moves to the LAST child position of *preccNewParent*. If *preccNewParent* is NULL, *preccMove* becomes the last root level record of the container.
- Other *preccMove* moves after this record in the list of children of *preccNewParent*. If *preccNewParent* is NULL, *preccMove* moves after the record specified by *pRecordOrder* only if that record is also a root level record.
- Note:** This record must currently exist in the list of children of *preccNewParent*.

TREEMOVE Field - flMoveSiblings

flMoveSiblings (BOOL)
Flag indicating whether to move siblings.

- TRUE All siblings of *preccMove* that FOLLOW it (from its original location) move to the new parent as well. *pRecordOrder* applies if this flag is TRUE.
- FALSE Only *preccMove* itself moves to the new parent; all siblings remain with the old parent.

UCHAR

Single-byte unsigned character or unsigned integer in the range 0 through 255.

```
typedef unsigned char UCHAR;
```

ULONG

32-bit unsigned integer in the range 0 through 4 294 967 295.

```
typedef unsigned long ULONG;
```

USERBUTTON

User-button data structure.

```
typedef struct _USERBUTTON {  
    HWND    hwnd;          /* Window handle. */  
    HPS     hps;           /* Presentation-space handle. */  
    ULONG    fsState;       /* New state of user button. */  
    ULONG    fsStateOld;    /* Old state of user button. */  
} USERBUTTON;  
  
typedef USERBUTTON *PUSERBUTTON;
```

USERBUTTON Field - hwnd

hwnd ([HWND](#))
Window handle.

USERBUTTON Field - hps

hps ([HPS](#))
Presentation-space handle.

USERBUTTON Field - fsState

fsState ([ULONG](#))
New state of user button.

USERBUTTON Field - fsStateOld

fsStateOld ([ULONG](#))
Old state of user button.

USHORT

Unsigned integer in the range 0 through 65 535.

```
typedef unsigned short USHORT;
```

VIDEOMODEINFO

The VIDEOMODEINFO data structure receives information for the current video monitor.

Note: The *cb* and *ulColors* fields are new to VIDEOMODEINFO. The color depth field (*ulColors*) was introduced to differentiate between pixel and color depth.

```
typedef struct _VIDEOMODEINFO {
    ULONG      cb; /* Size of the structure. */
    MODEID     miModeId; /* Used to make a SetMode request. */
    USHORT     usType; /* Flag indicating mode type. */
    USHORT     usInt10ModeSet; /* Interrupt 10 mode. */
    USHORT     usXResolution; /* Horizontal pixels. */
    USHORT     usYResolution; /* Vertical scanlines. */
    ULONG      ulBufferAddress; /* Physical address of VRAM. */
    ULONG      ulApertureSize; /* VRAM aperture. */
    ULONG      ulColors; /* Color depth. */
    BYTE       bBitsPerPixel; /* Pixel depth. */
    BYTE       bBitPlanes; /* Number of planes. */
    BYTE       bCharSize; /* Font width. */
    BYTE       bYCharSize; /* Font height. */
    USHORT     usBytesPerScanLine; /* Number of bytes per scan line. */
    USHORT     usTextRows; /* Number of text rows. */
    ULONG      ulPageLength; /* Number of bytes to save a plane. */
    ULONG      ulSaveSize; /* Total bytes of VRAM to save. */
    BYTE       bVrtRefresh; /* Vertical refresh rate. */
    BYTE       bHrtRefresh; /* Horizontal refresh rate. */
    BYTE       bVrtPolPos; /* Vertical polarity. */
    BYTE       bHrtPolPos; /* Horizontal polarity. */
    USHORT     usScrnTop; /* Vertical blanking away from the top, in line counts. */
    USHORT     usScrnBottom; /* Vertical blanking away from the bottom, in line counts. */
    USHORT     usScrnLeft; /* Horizontal blanking away from the left, in pixel counts. */
    USHORT     usScrnRight; /* Horizontal blanking away from the right, in pixel counts. */
    CHAR       szColorFormat[8]; /* Color format string for true color or high-color modes. */
}
```

```
    CHAR        szColorWeight[8];    /* Color weight string for true color or high-color modes. */
} VIDEOMODEINFO;

typedef VIDEOMODEINFO *FAR *PVIDEOMODEINFO;
```

VIDEOMODEINFO Field - cb

cb ([ULONG](#))
Size of the structure.

VIDEOMODEINFO Field - miModelId

miModelId (MODEID)
Used to make a SetMode request.

VIDEOMODEINFO Field - usType

usType ([USHORT](#))
Flag indicating mode type.

The following values are valid for this flag:

MODE_FLAG_NOT_MONO	0x0001; Mono-compatible
MODE_FLAG_GRAPHICS	0x0002; Text mode, Graphics
MODE_FLAG_NO_CLR_BRST	0x0004; Disable Color burst
MODE_FLAG_NATIVE	0x0008; Native (advanced function) mode
IGNORE_CLR_BRST	0x0010; Disable color burst; doesn't make sense for this mode
NOT_PLASMA	0x0020; will not work on plasma display
MODE_FLAG_VGA_ENTRY	0x0040; VGA mode, needs clean up

VIDEOMODEINFO Field - usInt10ModeSet

usInt10ModeSet ([USHORT](#))
Interrupt 10 mode.

VIDEOMODEINFO Field - usXResolution

usXResolution (USHORT)
Horizontal pixels.

VIDEOMODEINFO Field - usYResolution

usYResolution (USHORT)
Vertical scanlines.

VIDEOMODEINFO Field - ulBufferAddress

ulBufferAddress (ULONG)
Physical address of VRAM.

VIDEOMODEINFO Field - ulApertureSize

ulApertureSize (ULONG)
VRAM aperture.

VIDEOMODEINFO Field - ulColors

ulColors (ULONG)
Color depth.

VIDEOMODEINFO Field - bBitsPerPixel

bBitsPerPixel (BYTE)
Pixel depth.

VIDEOMODEINFO Field - bBitPlanes

bBitPlanes (BYTE)
Number of planes.

VIDEOMODEINFO Field - bXCharSize

bXCharSize (BYTE)
Font width.

VIDEOMODEINFO Field - bYCharSize

bYCharSize (BYTE)
Font height.

VIDEOMODEINFO Field - usBytesPerScanLine

usBytesPerScanLine (USHORT)
Number of bytes per scan line.

VIDEOMODEINFO Field - usTextRows

usTextRows (USHORT)
Number of text rows.

VIDEOMODEINFO Field - ulPageLength

ulPageLength (ULONG)
Number of bytes to save a plane.

VIDEOMODEINFO Field - ulSaveSize

ulSaveSize (ULONG)
Total bytes of VRAM to save.

VIDEOMODEINFO Field - bVrtRefresh

bVrtRefresh (BYTE)
Vertical refresh rate.

VIDEOMODEINFO Field - bHrtRefresh

bHrtRefresh (BYTE)
Horizontal refresh rate.

VIDEOMODEINFO Field - bVrtPolPos

bVrtPolPos (BYTE)
Vertical polarity.

VIDEOMODEINFO Field - bHrtPolPos

bHrtPolPos (BYTE)
Horizontal polarity.

VIDEOMODEINFO Field - usScrnTop

usScrnTop (USHORT)
Vertical blanking away from the top, in line counts.

VIDEOMODEINFO Field - usScrnBottom

usScrnBottom ([USHORT](#))

Vertical blanking away from the bottom, in line counts.

VIDEOMODEINFO Field - usScrnLeft

usScrnLeft ([USHORT](#))

Horizontal blanking away from the left, in pixel counts.

VIDEOMODEINFO Field - usScrnRight

usScrnRight ([USHORT](#))

Horizontal blanking away from the right, in pixel counts.

VIDEOMODEINFO Field - szColorFormat[8]

szColorFormat[8] ([CHAR](#))

Color format string for true color or high-color modes.

VIDEOMODEINFO Field - szColorWeight[8]

szColorWeight[8] ([CHAR](#))

Color weight string for true color or high-color modes.

VIOFONTCELLSIZE

VIO cell size. See [DevEscape](#).

```
typedef struct _VIOFONTCELLSIZE {  
    LONG    cx; /* Cell width. */  
};
```

```
    LONG    cy; /* Cell height. */
} VIOFONTCELLSIZE;

typedef VIOFONTCELLSIZE *PVIOFONTCELLSIZE;
```

VIOFONTCELLSIZE Field - cx

cx ([LONG](#))
Cell width.

VIOFONTCELLSIZE Field - cy

cy ([LONG](#))
Cell height.

VIOSIZECOUNT

Count of VIO cell sizes. See [DevEscape](#).

```
typedef struct _VIOSIZECOUNT {
    LONG    maxcount; /* Maximum number of VIO cell sizes supported. */
    LONG    count;    /* Number of VIO cell sizes returned. */
} VIOSIZECOUNT;

typedef VIOSIZECOUNT *PVIOSIZECOUNT;
```

VIOSIZECOUNT Field - maxcount

maxcount ([LONG](#))
Maximum number of VIO cell sizes supported.

VIOSIZECOUNT Field - count

count ([LONG](#))
Number of VIO cell sizes returned.

VOID

A data area of undefined format.

```
#define VOID void
```

VSCDATA

Structure that contains information about the value set control.

```
typedef struct _VSCDATA {  
    ULONG      cbSize;           /* Data length. */  
    USHORT     usRowCount;      /* Number of rows. */  
    USHORT     usColumnCount;   /* Number of columns. */  
} VSCDATA;  
  
typedef VSCDATA *PVSCDATA;
```

VSCDATA Field - cbSize

cbSize (ULONG)
Data length.

Length of the control data in bytes.

VSCDATA Field - usRowCount

usRowCount (USHORT)
Number of rows.

The number of rows in the value set control. The minimum number of rows is 1 and the maximum number of rows is 65,535.

VSCDATA Field - usColumnCount

usColumnCount (USHORT)
Number of columns.

The number of columns in the value set control. The minimum number of columns is 1 and the maximum number of columns is

65,535.

VSDRAGINFO

Structure that contains information about direct manipulation actions that occur over the value set control.

```
typedef struct _VSDRAGINFO {  
    PDRAGINFO    pDragInfo; /* Pointer to a DRAGINFO structure. */  
    USHORT       usRow;     /* Row index. */  
    USHORT       usColumn;  /* Column index. */  
} VSDRAGINFO;  
  
typedef VSDRAGINFO *PVSDRAGINFO;
```

VSDRAGINFO Field - pDragInfo

pDragInfo ([PDRAGINFO](#))
Pointer to a [DRAGINFO](#) structure.

VSDRAGINFO Field - usRow

usRow ([USHORT](#))
Row index.

The index of the row over which the direct manipulation action occurred.

VSDRAGINFO Field - usColumn

usColumn ([USHORT](#))
Column index.

The index of the column over which the direct manipulation action occurred.

VSDRAGINIT

Structure that contains information that is used to initialize a direct manipulation action over the value set control.

```
typedef struct _VSDRAGINIT {
```

```
HWND      hwnd;      /* Value set window handle. */
LONG      x;          /* X-coordinate. */
LONG      y;          /* Y-coordinate. */
LONG      cx;         /* X-offset. */
LONG      cy;         /* Y-offset. */
USHORT    usRow;      /* Row index. */
USHORT    usColumn;   /* Column index. */
} VSDRAGINIT;

typedef VSDRAGINIT *PVSDRAGINIT;
```

VSDRAGINIT Field - hwnd

hwnd ([HWND](#))
Value set window handle.

Window handle of the value set control.

VSDRAGINIT Field - x

x ([LONG](#))
X-coordinate.

X-coordinate of the pointing device pointer in desktop coordinates.

VSDRAGINIT Field - y

y ([LONG](#))
Y-coordinate.

Y-coordinate of the pointing device pointer in desktop coordinates.

VSDRAGINIT Field - cx

cx ([LONG](#))
X-offset.

X-offset from the hot spot of the pointing device pointer, in pels, to the item origin. The item origin is the lower left corner of the item.

VSDRAGINIT Field - cy

cy ([LONG](#))
Y-offset.

Y-offset from the hot spot of the pointing device pointer, in pels, to the item origin. The item origin is the lower left corner of the item.

VSDRAGINIT Field - usRow

usRow ([USHORT](#))
Row index.

The index of the row over which the direct manipulation action occurred.

VSDRAGINIT Field - usColumn

usColumn ([USHORT](#))
Column index.

The index of the column over which the direct manipulation action occurred.

VSTEXT

Value set text structure. This structure is used with the [VM_QUERYITEM](#) message only.

```
typedef struct _VSTEXT {  
    PSZ      pszItemText; /* Pointer to a buffer to copy the string into. */  
    ULONG    ulBufLen;    /* Buffer size. */  
} VSTEXT;  
  
typedef VSTEXT *PVSTEXT;
```

VSTEXT Field - pszItemText

pszItemText ([PSZ](#))
Pointer to a buffer to copy the string into.

VSTEXT Field - ulBufLen

ulBufLen (ULONG)
Buffer size.

Size of the buffer pointed to by the *pszItemText* field.

WNDPARAMS

Window parameters.

```
typedef struct _WNDPARAMS {
    ULONG    fsStatus;      /* Window parameter selection. */
    ULONG    cchText;      /* Length of window text. */
    PSZ      pszText;      /* Window text. */
    ULONG    cbPresParams; /* Length of presentation parameters. */
    PVOID    pPresParams;  /* Presentation parameters. */
    ULONG    cbCtlData;    /* Length of window class specific data. */
    PVOID    pCtlData;     /* Window class specific data. */
} WNDPARAMS;

typedef WNDPARAMS *PWNDPARAMS;
```

WNDPARAMS Field - fsStatus

fsStatus (ULONG)
Window parameter selection.

Identifies the window parameters that are to be set or queried:

- WPM_CBCTLDATA
Window control data length
- WPM_CCHTEXT
Window text length
- WPM_CTLDATA
Window control data
- WPM_PRESPARAMS
Presentation parameters
- WPM_TEXT
Window text.

WNDPARAMS Field - cchText

cchText (ULONG)
Length of window text.

WNDPARAMS Field - pszText

pszText ([PSZ](#))
Window text.

WNDPARAMS Field - cbPresParams

cbPresParams ([ULONG](#))
Length of presentation parameters.

WNDPARAMS Field - pPresParams

pPresParams ([PVOID](#))
Presentation parameters.

WNDPARAMS Field - cbCtlData

cbCtlData ([ULONG](#))
Length of window class specific data.

WNDPARAMS Field - pCtlData

pCtlData ([PVOID](#))
Window class specific data.

WPOINT

Window-point data structure (long integers). See [POINTL](#) for the form of the structure.

```
#define WPOINT POINTL
```

WRECT

Window-rectangle data structure. See [RECTL](#) for the form of the structure.

```
#define WRECT RECTL
```

XYWINSIZE

Window position and size structure.

```
typedef struct _XYWINSIZE {
    SHORT      x;          /* X-coordinate of window origin. */
    SHORT      y;          /* Y-coordinate of window origin. */
    SHORT      cx;         /* Window width. */
    SHORT      cy;         /* Window height. */
    USHORT     fsWindow;   /* Window options. */
} XYWINSIZE;

typedef XYWINSIZE *PXYWINSIZE;
```

XYWINSIZE Field - x

x ([SHORT](#))
X-coordinate of window origin.

XYWINSIZE Field - y

y ([SHORT](#))
Y-coordinate of window origin.

XYWINSIZE Field - cx

cx ([SHORT](#))
Window width.

XYWINSIZE Field - cy

cy ([SHORT](#))
Window height.

XYWINSIZE Field - fsWindow

fsWindow ([USHORT](#))
Window options.

The values may be ORed together. For example, an invisible iconic window can be created. Note that if both XYF_MINIMIZED and XYF_MAXIMIZED are specified, the window is created in a maximized state.

- XYF_INVISIBLE
Create the window initially invisible.
- XYF_MAXIMIZED
Show the window initially maximized.
- XYF_MINIMIZED
Show the window initially iconic.
- XYF_NOAUTOCLOSE
Do not close the window automatically when the VIO application terminates. This parameter is ignored unless the program is a VIO-windowed application.
- XYF_NORMAL
Create the window visible, with a size and position as specified. This is the default.

Errors

Error codes are furnished numerically and alphabetically.

For a listing of error codes by number, see [Error Number and Name](#).

For a listing of error codes and their explanations, see [Error Name and Explanation](#).

Error Number and Name

This section lists PM errors returned by WinGetLastError in order of their error numbers. For explanations of these errors, see [Error Name and Explanation](#).

Error Number	Error Constant
0x0000	PMERR_OK
0x0836	NERR_NetNotStarted
0x0845	NERR_RedirectedPath
0x084B	NERR_BufTooSmall
0x085E	NERR_InvalidAPI
0x0866	NERR_QNotFound
0x0867	NERR_JobNotFound
0x0868	NERR_DestNotFound
0x0869	NERR_DestExists
0x086A	NERR_QExists
0x086B	NERR_QNoRoom
0x086C	NERR_JobNoRoom
0x086D	NERR_DestNoRoom
0x086E	NERR_DestIdle
0x086F	NERR_DestInvalidOp
0x0871	NERR_SpoolerNotLoaded
0x0872	NERR_DestInvalidState

0x0874	NERR_JobInvalidState
0x0875	NERR_SpoolNoMemory
0x0876	NERR_DriverNotFound
0x0877	NERR_DataTypeInvalid
0x0878	NERR_ProcNotFound
0x0925	NERR_BadDev
0x0927	NERR_CommDevInUse
0x092F	NERR_InvalidComputer
0x0961	NERR_OpenFiles
0x0965	NERR_LocalDrive
0x1001	PMERR_INVALID_HWND
0x1001	HMERR_NO_FRAME_WND_IN_CHAIN
0x1002	PMERR_INVALID_HMQ
0x1002	HMERR_INVALID_ASSOC_APP_WND
0x1003	PMERR_PARAMETER_OUT_OF_RANGE
0x1003	HMERR_INVALID_ASSOC_HELP_INST
0x1004	PMERR_WINDOW_LOCK_UNDERFLOW
0x1004	HMERR_INVALID_DESTROY_HELP_INST
0x1005	PMERR_WINDOW_LOCK_OVERFLOW
0x1005	HMERR_NO_HELP_INST_IN_CHAIN
0x1006	PMERR_BAD_WINDOW_LOCK_COUNT
0x1006	HMERR_INVALID_HELP_INSTANCE_HDL
0x1007	PMERR_WINDOW_NOT_LOCKED
0x1007	HMERR_INVALID_QUERY_APP_WND
0x1008	PMERR_INVALID_SELECTOR
0x1008	HMERR_HELP_INST_CALLED_INVALID
0x1009	PMERR_CALL_FROM_WRONG_THREAD
0x1009	HMERR_HELP_TABLE_UNDEFINE
0x100A	PMERR_RESOURCE_NOT_FOUND
0x100A	HMERR_HELP_INSTANCE_UNDEFINE
0x100B	PMERR_INVALID_STRING_PARM
0x100B	HMERR_HELP_ITEM_NOT_FOUND
0x100C	PMERR_INVALID_HHEAP
0x100C	HMERR_INVALID_HELP_SUBITEM_SIZE
0x100D	PMERR_INVALID_HEAP_POINTER
0x100D	HMERR_HELP_SUBITEM_NOT_FOUND
0x100E	PMERR_INVALID_HEAP_SIZE_PARM
0x100F	PMERR_INVALID_HEAP_SIZE
0x1010	PMERR_INVALID_HEAP_SIZE_WORD
0x1011	PMERR_HEAP_OUT_OF_MEMORY
0x1012	PMERR_HEAP_MAX_SIZE_REACHED
0x1013	PMERR_INVALID_HATOM_TBL
0x1014	PMERR_INVALID_ATOM
0x1015	PMERR_INVALID_ATOM_NAME
0x1016	PMERR_INVALID_INTEGER_ATOM
0x1017	PMERR_ATOM_NAME_NOT_FOUND
0x1018	PMERR_QUEUE_TOO_LARGE
0x1019	PMERR_INVALID_FLAG
0x101A	PMERR_INVALID_HACCEL
0x101B	PMERR_INVALID_HPTR
0x101C	PMERR_INVALID_HENUM
0x101D	PMERR_INVALID_SRC_CODEPAGE
0x101E	PMERR_INVALID_DST_CODEPAGE
0x101F	PMERR_UNKNOWN_COMPONENT_ID
0x1020	PMERR_UNKNOWN_ERROR_CODE
0x1021	PMERR_SEVERITY_LEVELS
0x1034	PMERR_INVALID_RESOURCE_FORMAT
0x1035	WINDBG_WINDOW_UNLOCK_WAIT
0x1036	PMERR_NO_MSG_QUEUE
0x1037	PMERR_CANNOT_SET_FOCUS
0x1038	PMERR_QUEUE_FULL
0x1039	PMERR_LIBRARY_LOAD_FAILED
0x103A	PMERR_PROCEDURE_LOAD_FAILED
0x103B	PMERR_LIBRARY_DELETE_FAILED
0x103C	PMERR_PROCEDURE_DELETE_FAILED
0x103D	PMERR_ARRAY_TOO_LARGE
0x103E	PMERR_ARRAY_TOO_SMALL
0x103F	PMERR_DATATYPE_ENTRY_BAD_INDEX
0x1040	PMERR_DATATYPE_ENTRY_CTL_BAD
0x1041	PMERR_DATATYPE_ENTRY_CTL_MISS
0x1042	PMERR_DATATYPE_ENTRY_INVALID
0x1043	PMERR_DATATYPE_ENTRY_NOT_NUM
0x1044	PMERR_DATATYPE_ENTRY_NOT_OFF

0x1045	PMERR_DATATYPE_INVALID
0x1046	PMERR_DATATYPE_NOT_UNIQUE
0x1047	PMERR_DATATYPE_TOO_LONG
0x1048	PMERR_DATATYPE_TOO_SMALL
0x1049	PMERR_DIRECTION_INVALID
0x104A	PMERR_INVALID_HAB
0x104D	PMERR_INVALID_HSTRUCT
0x104E	PMERR_LENGTH_TOO_SMALL
0x104F	PMERR_MSGID_TOO_SMALL
0x1050	PMERR_NO_HANDLE_ALLOC
0x1051	PMERR_NOT_IN_A_PM_SESSION
0x1052	PMERR_MSG_QUEUE_ALREADY_EXISTS
0x1055	PMERR_OLD_RESOURCE
0x1056	PMERR_WPDSEVER_IS_ACTIVE
0x1057	PMERR_WPDSEVER_NOT_STARTED
0x1058	PMERR_SOMDD_IS_ACTIVE
0x1059	PMERR_SOMDD_NOT_STARTED
0x1101	PMERR_INVALID_PIB
0x1102	PMERR_INSUFF_SPACE_TO_ADD
0x1103	PMERR_INVALID_GROUP_HANDLE
0x1104	PMERR_DUPLICATE_TITLE
0x1105	PMERR_INVALID_TITLE
0x1106	PMERR_INVALID_TARGET_HANDLE
0x1107	PMERR_HANDLE_NOT_IN_GROUP
0x1108	PMERR_INVALID_PATH_STATEMENT
0x1109	PMERR_NO_PROGRAM_FOUND
0x110A	PMERR_INVALID_BUFFER_SIZE
0x110B	PMERR_BUFFER_TOO_SMALL
0x110C	PMERR_PL_INITIALIZATION_FAIL
0x110D	PMERR_CANT_DESTROY_SYS_GROUP
0x110E	PMERR_INVALID_TYPE_CHANGE
0x110F	PMERR_INVALID_PROGRAM_HANDLE
0x1110	PMERR_NOT_CURRENT_PL_VERSION
0x1111	PMERR_INVALID_CIRCULAR_REF
0x1112	PMERR_MEMORY_ALLOCATION_ERR
0x1113	PMERR_MEMORY_DEALLOCATION_ERR
0x1114	PMERR_TASK_HEADER_TOO_BIG
0x1115	PMERR_INVALID_INI_FILE_HANDLE
0x1116	PMERR_MEMORY_SHARE
0x1117	PMERR_OPEN_QUEUE
0x1118	PMERR_CREATE_QUEUE
0x1119	PMERR_WRITE_QUEUE
0x111A	PMERR_READ_QUEUE
0x111B	PMERR_CALL_NOT_EXECUTED
0x111C	PMERR_UNKNOWN_APIKT
0x111D	PMERR_INITHREAD_EXISTS
0x111E	PMERR_CREATE_THREAD
0x111F	PMERR_NO_HK_PROFILE_INSTALLED
0x1120	PMERR_INVALID_DIRECTORY
0x1121	PMERR_WILDCARD_IN_FILENAME
0x1122	PMERR_FILENAME_BUFFER_FULL
0x1123	PMERR_FILENAME_TOO_LONG
0x1124	PMERR_INI_FILE_IS_SYS_OR_USER
0x1125	PMERR_BROADCAST_PLMSG
0x1126	PMERR_190_INIT_DONE
0x1127	PMERR_HMOD_FOR_PMSHAPI
0x1128	PMERR_SET_HK_PROFILE
0x1129	PMERR_API_NOT_ALLOWED
0x112A	PMERR_INI_STILL_OPEN
0x112B	PMERR_PROGDETAILS_NOT_IN_INI
0x112C	PMERR_PIBSTRUCT_NOT_IN_INI
0x112D	PMERR_INVALID_DISKPROGDETAILS
0x112E	PMERR_PROGDETAILS_READ_FAILURE
0x112F	PMERR_PROGDETAILS_WRITE_FAILURE
0x1130	PMERR_PROGDETAILS_QSIZE_FAILURE
0x1131	PMERR_INVALID_PROGDETAILS
0x1132	PMERR_SHEPROFILEHOOK_NOT_FOUND
0x1133	PMERR_190PLCONVERTED
0x1134	PMERR_FAILED_TO_CONVERT_INI_PL
0x1135	PMERR_PMSHAPI_NOT_INITIALISED
0x1136	PMERR_INVALID_SHELL_API_HOOK_ID
0x1200	PMERR_DOS_ERROR
0x1201	PMERR_NO_SPACE

0x1202	PMERR_INVALID_SWITCH_HANDLE
0x1203	PMERR_NO_HANDLE
0x1204	PMERR_INVALID_PROCESS_ID
0x1205	PMERR_NOT_SHELL
0x1206	PMERR_INVALID_WINDOW
0x1207	PMERR_INVALID_POST_MSG
0x1208	PMERR_INVALID_PARAMETERS
0x1208	PMERR_INVALID_PARAMETERS
0x1209	PMERR_INVALID_PROGRAM_TYPE
0x120A	PMERR_NOT_EXTENDED_FOCUS
0x120B	PMERR_INVALID_SESSION_ID
0x120C	PMERR_SMG_INVALID_ICON_FILE
0x120D	PMERR_SMG_ICON_NOT_CREATED
0x120E	PMERR_SHL_DEBUG
0x1301	PMERR_OPENING_INI_FILE
0x1302	PMERR_INI_FILE_CORRUPT
0x1303	PMERR_INVALID_PARM
0x1304	PMERR_NOT_IN_IDX
0x1305	PMERR_NO_ENTRIES_IN_GROUP
0x1306	PMERR_INI_WRITE_FAIL
0x1307	PMERR_IDX_FULL
0x1308	PMERR_INI_PROTECTED
0x1309	PMERR_MEMORY_ALLOC
0x130A	PMERR_INI_INIT_ALREADY_DONE
0x130B	PMERR_INVALID_INTEGER
0x130C	PMERR_INVALID_ASCIZ
0x130D	PMERR_CAN_NOT_CALL_SPOOLER
0x130D	PMERR_VALIDATION_REJECTED
0x1401	PMERR_WARNING_WINDOW_NOT_KILLED
0x1402	PMERR_ERROR_INVALID_WINDOW
0x1403	PMERR_ALREADY_INITIALIZED
0x1405	PMERR_MSG_PROG_NO_MOU
0x1406	PMERR_MSG_PROG_NON_RECOV
0x1407	PMERR_WINCONV_INVALID_PATH
0x1408	PMERR_PI_NOT_INITIALISED
0x1409	PMERR_PL_NOT_INITIALISED
0x140A	PMERR_NO_TASK_MANAGER
0x140B	PMERR_SAVE_NOT_IN_PROGRESS
0x140C	PMERR_NO_STACK_SPACE
0x140D	PMERR_INVALID_COLR_FIELD
0x140E	PMERR_INVALID_COLR_VALUE
0x140F	PMERR_COLR_WRITE
0x1501	PMERR_TARGET_FILE_EXISTS
0x1502	PMERR_SOURCE_SAME_AS_TARGET
0x1503	PMERR_SOURCE_FILE_NOT_FOUND
0x1504	PMERR_INVALID_NEW_PATH
0x1505	PMERR_TARGET_FILE_NOT_FOUND
0x1506	PMERR_INVALID_DRIVE_NUMBER
0x1507	PMERR_NAME_TOO_LONG
0x1508	PMERR_NOT_ENOUGH_ROOM_ON_DISK
0x1509	PMERR_NOT_ENOUGH_MEM
0x150B	PMERR_LOG_DRV_DOES_NOT_EXIST
0x150C	PMERR_INVALID_DRIVE
0x150D	PMERR_ACCESS_DENIED
0x150E	PMERR_NO_FIRST_SLASH
0x150F	PMERR_READ_ONLY_FILE
0x151F	PMERR_GROUP_PROTECTED
0x152F	PMERR_INVALID_PROGRAM_CATEGORY
0x1530	PMERR_INVALID_APPL
0x1531	PMERR_CANNOT_START
0x1532	PMERR_STARTED_IN_BACKGROUND
0x1533	PMERR_INVALID_HAPP
0x1534	PMERR_CANNOT_STOP
0x1601	PMERR_INTERNAL_ERROR_1
0x1602	PMERR_INTERNAL_ERROR_2
0x1603	PMERR_INTERNAL_ERROR_3
0x1604	PMERR_INTERNAL_ERROR_4
0x1605	PMERR_INTERNAL_ERROR_5
0x1606	PMERR_INTERNAL_ERROR_6
0x1607	PMERR_INTERNAL_ERROR_7
0x1608	PMERR_INTERNAL_ERROR_8
0x1609	PMERR_INTERNAL_ERROR_9
0x160A	PMERR_INTERNAL_ERROR_10

0x160B	PMERR_INTERNAL_ERROR_11
0x160C	PMERR_INTERNAL_ERROR_12
0x160D	PMERR_INTERNAL_ERROR_13
0x160E	PMERR_INTERNAL_ERROR_14
0x160F	PMERR_INTERNAL_ERROR_15
0x1610	PMERR_INTERNAL_ERROR_16
0x1611	PMERR_INTERNAL_ERROR_17
0x1612	PMERR_INTERNAL_ERROR_18
0x1613	PMERR_INTERNAL_ERROR_19
0x1614	PMERR_INTERNAL_ERROR_20
0x1615	PMERR_INTERNAL_ERROR_21
0x1616	PMERR_INTERNAL_ERROR_22
0x1617	PMERR_INTERNAL_ERROR_23
0x1618	PMERR_INTERNAL_ERROR_24
0x1619	PMERR_INTERNAL_ERROR_25
0x161A	PMERR_INTERNAL_ERROR_26
0x161B	PMERR_INTERNAL_ERROR_27
0x161C	PMERR_INTERNAL_ERROR_28
0x161D	PMERR_INTERNAL_ERROR_29
0x1630	PMERR_INVALID_FREE_MESSAGE_ID
0x1641	PMERR_FUNCTION_NOT_SUPPORTED
0x1642	PMERR_INVALID_ARRAY_COUNT
0x1643	PMERR_INVALID_LENGTH
0x1644	PMERR_INVALID_BUNDLE_TYPE
0x1645	PMERR_INVALID_PARAMETER
0x1646	PMERR_INVALID_NUMBER_OF_PARMS
0x1647	PMERR_GREATER_THAN_64K
0x1648	PMERR_INVALID_PARAMETER_TYPE
0x1649	PMERR_NEGATIVE_STRCOND_DIM
0x164A	PMERR_INVALID_NUMBER_OF_TYPES
0x164B	PMERR_INCORRECT_HSTRUCT
0x164C	PMERR_INVALID_ARRAY_SIZE
0x164D	PMERR_INVALID_CONTROL_DATATYPE
0x164E	PMERR_INCOMPLETE_CONTROL_SEQU
0x164F	PMERR_INVALID_DATATYPE
0x1650	PMERR_INCORRECT_DATATYPE
0x1651	PMERR_NOT_SELF_DESCRIBING_DTYP
0x1652	PMERR_INVALID_CTRL_SEQ_INDEX
0x1653	PMERR_INVALID_TYPE_FOR_LENGTH
0x1654	PMERR_INVALID_TYPE_FOR_OFFSET
0x1655	PMERR_INVALID_TYPE_FOR_MPARAM
0x1656	PMERR_INVALID_MESSAGE_ID
0x1657	PMERR_C_LENGTH_TOO_SMALL
0x1658	PMERR_APPL_STRUCTURE_TOO_SMALL
0x1659	PMERR_INVALID_ERRORINFO_HANDLE
0x165A	PMERR_INVALID_CHARACTER_INDEX
0x1700	WPERR_PROTECTED_CLASS
0x1701	WPERR_INVALID_CLASS
0x1702	WPERR_INVALID_SUPERCLASS
0x1703	WPERR_NO_MEMORY
0x1704	WPERR_SEMAPHORE_ERROR
0x1705	WPERR_BUFFER_TOO_SMALL
0x1706	WPERR_CLSLOADMOD_FAILED
0x1707	WPERR_CLSPROCADDR_FAILED
0x1708	WPERR_OBJWORD_LOCATION
0x1709	WPERR_INVALID_OBJECT
0x170A	WPERR_MEMORY_CLEANUP
0x170B	WPERR_INVALID_MODULE
0x170C	WPERR_INVALID_OLDCLASS
0x170D	WPERR_INVALID_NEWCLASS
0x170E	WPERR_NOT_IMMEDIATE_CHILD
0x170F	WPERR_NOT_WORKPLACE_CLASS
0x1710	WPERR_CANT_REPLACE_METACLAS
0x1711	WPERR_INI_FILE_WRITE
0x1712	WPERR_INVALID_FOLDER
0x1713	WPERR_BUFFER_OVERFLOW
0x1714	WPERR_OBJECT_NOT_FOUND
0x1715	WPERR_INVALID_HFIND
0x1716	WPERR_INVALID_COUNT
0x1717	WPERR_INVALID_BUFFER
0x1718	WPERR_ALREADY_EXISTS
0x1719	WPERR_INVALID_FLAGS
0x1720	WPERR_INVALID_OBJECTID

0x1721	WPERR_INVALID_TARGET_OBJECT
0x1F00	PMERR_NOT_DRAGGING
0x2001	PMERR_ALREADY_IN_AREA
0x2001	HMERR_INDEX_NOT_FOUND
0x2002	PMERR_ALREADY_IN_ELEMENT
0x2002	HMERR_CONTENT_NOT_FOUND
0x2003	PMERR_ALREADY_IN_PATH
0x2003	HMERR_OPEN_LIB_FILE
0x2004	PMERR_ALREADY_IN_SEG
0x2004	HMERR_READ_LIB_FILE
0x2005	PMERR_AREA_INCOMPLETE
0x2005	HMERR_CLOSE_LIB_FILE
0x2006	PMERR_BASE_ERROR
0x2006	HMERR_INVALID_LIB_FILE
0x2007	PMERR_BITBLT_LENGTH_EXCEEDED
0x2007	HMERR_NO_MEMORY
0x2008	PMERR_BITMAP_IN_USE
0x2008	HMERR_ALLOCATE_SEGMENT
0x2009	PMERR_BITMAP_IS_SELECTED
0x2009	HMERR_FREE_MEMORY
0x200A	PMERR_BITMAP_NOT_FOUND
0x200B	PMERR_BITMAP_NOT_SELECTED
0x200C	PMERR_BOUNDS_OVERFLOW
0x200D	PMERR_CALLED_SEG_IS_CHAINED
0x200E	PMERR_CALLED_SEG_IS_CURRENT
0x200F	PMERR_CALLED_SEG_NOT_FOUND
0x2010	PMERR_CANNOT_DELETE_ALL_DATA
0x2010	HMERR_PANEL_NOT_FOUND
0x2011	PMERR_CANNOT_REPLACE_ELEMENT_0
0x2011	HMERR_DATABASE_NOT_OPEN
0x2012	PMERR_COL_TABLE_NOT_REALIZABLE
0x2013	PMERR_COL_TABLE_NOT_REALIZED
0x2013	HMERR_LOAD_DLL
0x2014	PMERR_COORDINATE_OVERFLOW
0x2015	PMERR_CORR_FORMAT_MISMATCH
0x2016	PMERR_DATA_TOO_LONG
0x2017	PMERR_DC_IS_ASSOCIATED
0x2018	PMERR_DESC_STRING_TRUNCATED
0x2019	PMERR_DEVICE_DRIVER_ERROR_1
0x201A	PMERR_DEVICE_DRIVER_ERROR_2
0x201B	PMERR_DEVICE_DRIVER_ERROR_3
0x201C	PMERR_DEVICE_DRIVER_ERROR_4
0x201D	PMERR_DEVICE_DRIVER_ERROR_5
0x201E	PMERR_DEVICE_DRIVER_ERROR_6
0x201F	PMERR_DEVICE_DRIVER_ERROR_7
0x2020	PMERR_DEVICE_DRIVER_ERROR_8
0x2021	PMERR_DEVICE_DRIVER_ERROR_9
0x2022	PMERR_DEVICE_DRIVER_ERROR_10
0x2023	PMERR_DEV_FUNC_NOT_INSTALLED
0x2024	PMERR_DOSOPEN_FAILURE
0x2025	PMERR_DOSREAD_FAILURE
0x2026	PMERR_DRIVER_NOT_FOUND
0x2027	PMERR_DUP_SEG
0x2028	PMERR_DYNAMIC_SEG_SEQ_ERROR
0x2029	PMERR_DYNAMIC_SEG_ZERO_INV
0x202A	PMERR_ELEMENT_INCOMPLETE
0x202B	PMERR_ESC_CODE_NOT_SUPPORTED
0x202C	PMERR_EXCEEDS_MAX_SEG_LENGTH
0x202D	PMERR_FONT_AND_MODE_MISMATCH
0x202E	PMERR_FONT_FILE_NOT_LOADED
0x202F	PMERR_FONT_NOT_LOADED
0x2030	PMERR_FONT_TOO_BIG
0x2031	PMERR_HARDWARE_INIT_FAILURE
0x2032	PMERR_HBITMAP_BUSY
0x2033	PMERR_HDC_BUSY
0x2034	PMERR_HRGN_BUSY
0x2035	PMERR_HUGE_FONTS_NOT_SUPPORTED
0x2036	PMERR_ID_HAS_NO_BITMAP
0x2037	PMERR_IMAGE_INCOMPLETE
0x2038	PMERR_INCOMPAT_COLOR_FORMAT
0x2039	PMERR_INCOMPAT_COLOR_OPTIONS
0x203A	PMERR_INCOMPATIBLE_BITMAP
0x203B	PMERR_INCOMPATIBLE_METAFILE

0x203C	PMERR_INCORRECT_DC_TYPE
0x203D	PMERR_INSUFFICIENT_DISK_SPACE
0x203E	PMERR_INSUFFICIENT_MEMORY
0x203F	PMERR_INV_ANGLE_PARM
0x2040	PMERR_INV_ARC_CONTROL
0x2041	PMERR_INV_AREA_CONTROL
0x2042	PMERR_INV_ARC_POINTS
0x2043	PMERR_INV_ATTR_MODE
0x2044	PMERR_INV_BACKGROUND_COL_ATTR
0x2045	PMERR_INV_BACKGROUND_MIX_ATTR
0x2046	PMERR_INV_BITBLT_MIX
0x2047	PMERR_INV_BITBLT_STYLE
0x2048	PMERR_INV_BITMAP_DIMENSION
0x2049	PMERR_INV_BOX_CONTROL
0x204A	PMERR_INV_BOX_ROUNDING_PARM
0x204B	PMERR_INV_CHAR_ANGLE_ATTR
0x204C	PMERR_INV_CHAR_DIRECTION_ATTR
0x204D	PMERR_INV_CHAR_MODE_ATTR
0x204E	PMERR_INV_CHAR_POS_OPTIONS
0x204F	PMERR_INV_CHAR_SET_ATTR
0x2050	PMERR_INV_CHAR_SHEAR_ATTR
0x2051	PMERR_INV_CLIP_PATH_OPTIONS
0x2052	PMERR_INV_CODEPAGE
0x2053	PMERR_INV_COLOR_ATTR
0x2054	PMERR_INV_COLOR_DATA
0x2055	PMERR_INV_COLOR_FORMAT
0x2056	PMERR_INV_COLOR_INDEX
0x2057	PMERR_INV_COLOR_OPTIONS
0x2058	PMERR_INV_COLOR_START_INDEX
0x2059	PMERR_INV_COORD_OFFSET
0x205A	PMERR_INV_COORD_SPACE
0x205B	PMERR_INV_COORDINATE
0x205C	PMERR_INV_CORRELATE_DEPTH
0x205D	PMERR_INV_CORRELATE_TYPE
0x205E	PMERR_INV_CURSOR_BITMAP
0x205F	PMERR_INV_DC_DATA
0x2060	PMERR_INV_DC_TYPE
0x2061	PMERR_INV_DEVICE_NAME
0x2062	PMERR_INV_DEV_MODES_OPTIONS
0x2063	PMERR_INV_DRAW_CONTROL
0x2064	PMERR_INV_DRAW_VALUE
0x2065	PMERR_INV_DRAWING_MODE
0x2066	PMERR_INV_DRIVER_DATA
0x2067	PMERR_INV_DRIVER_NAME
0x2068	PMERR_INV_DRAW_BORDER_OPTION
0x2069	PMERR_INV_EDIT_MODE
0x206A	PMERR_INV_ELEMENT_OFFSET
0x206B	PMERR_INV_ELEMENT_POINTER
0x206C	PMERR_INV_END_PATH_OPTIONS
0x206D	PMERR_INV_ESC_CODE
0x206E	PMERR_INV_ESCAPE_DATA
0x206F	PMERR_INV_EXTENDED_LCID
0x2070	PMERR_INV_FILL_PATH_OPTIONS
0x2071	PMERR_INV_FIRST_CHAR
0x2072	PMERR_INV_FONT_ATTRS
0x2073	PMERR_INV_FONT_FILE_DATA
0x2074	PMERR_INV_FOR_THIS_DC_TYPE
0x2075	PMERR_INV_FORMAT_CONTROL
0x2076	PMERR_INV_FORMS_CODE
0x2077	PMERR_INV_FONTDEF
0x2078	PMERR_INV_GEOM_LINE_WIDTH_ATTR
0x2079	PMERR_INV_GETDATA_CONTROL
0x207A	PMERR_INV_GRAPHICS_FIELD
0x207B	PMERR_INV_HBITMAP
0x207C	PMERR_INV_HDC
0x207D	PMERR_INV_HJOURNAL
0x207E	PMERR_INV_HMF
0x207F	PMERR_INV_HPS
0x2080	PMERR_INV_HRGN
0x2081	PMERR_INV_ID
0x2082	PMERR_INV_IMAGE_DATA_LENGTH
0x2083	PMERR_INV_IMAGE_DIMENSION
0x2084	PMERR_INV_IMAGE_FORMAT

0x2085	PMERR_INV_IN_AREA
0x2086	PMERR_INV_IN_CALLED_SEG
0x2087	PMERR_INV_IN_CURRENT_EDIT_MODE
0x2088	PMERR_INV_IN_DRAW_MODE
0x2089	PMERR_INV_IN_ELEMENT
0x208A	PMERR_INV_IN_IMAGE
0x208B	PMERR_INV_IN_PATH
0x208C	PMERR_INV_IN_RETAIN_MODE
0x208D	PMERR_INV_IN_SEG
0x208E	PMERR_INV_IN_VECTOR_SYMBOL
0x208F	PMERR_INV_INFO_TABLE
0x2090	PMERR_INV_JOURNAL_OPTION
0x2091	PMERR_INV_KERNING_FLAGS
0x2092	PMERR_INV_LENGTH_OR_COUNT
0x2093	PMERR_INV_LINE_END_ATTR
0x2094	PMERR_INV_LINE_JOIN_ATTR
0x2095	PMERR_INV_LINE_TYPE_ATTR
0x2096	PMERR_INV_LINE_WIDTH_ATTR
0x2097	PMERR_INV_LOGICAL_ADDRESS
0x2098	PMERR_INV_MARKER_BOX_ATTR
0x2099	PMERR_INV_MARKER_SET_ATTR
0x209A	PMERR_INV_MARKER_SYMBOL_ATTR
0x209B	PMERR_INV_MATRIX_ELEMENT
0x209C	PMERR_INV_MAX_HITS
0x209D	PMERR_INV_METAFILE
0x209E	PMERR_INV_METAFILE_LENGTH
0x209F	PMERR_INV_METAFILE_OFFSET
0x20A0	PMERR_INV_MICROPS_DRAW_CONTROL
0x20A1	PMERR_INV_MICROPS_FUNCTION
0x20A2	PMERR_INV_MICROPS_ORDER
0x20A3	PMERR_INV_MIX_ATTR
0x20A4	PMERR_INV_MODE_FOR_OPEN_DYN
0x20A5	PMERR_INV_MODE_FOR_REOPEN_SEG
0x20A6	PMERR_INV_MODIFY_PATH_MODE
0x20A7	PMERR_INV_MULTIPLIER
0x20A8	PMERR_INV_NESTED_FIGURES
0x20A9	PMERR_INV_OR_INCOMPAT_OPTIONS
0x20AA	PMERR_INV_ORDER_LENGTH
0x20AB	PMERR_INV_ORDERING_PARM
0x20AC	PMERR_INV_OUTSIDE_DRAW_MODE
0x20AD	PMERR_INV_PAGE_VIEWPORT
0x20AE	PMERR_INV_PATH_ID
0x20AF	PMERR_INV_PATH_MODE
0x20B0	PMERR_INV_PATTERN_ATTR
0x20B1	PMERR_INV_PATTERN_REF_PT_ATTR
0x20B2	PMERR_INV_PATTERN_SET_ATTR
0x20B3	PMERR_INV_PATTERN_SET_FONT
0x20B4	PMERR_INV_PICK_APERTURE_OPTION
0x20B5	PMERR_INV_PICK_APERTURE_POSN
0x20B6	PMERR_INV_PICK_APERTURE_SIZE
0x20B7	PMERR_INV_PICK_NUMBER
0x20B8	PMERR_INV_PLAY_METAFILE_OPTION
0x20B9	PMERR_INV_PRIMITIVE_TYPE
0x20BA	PMERR_INV_PS_SIZE
0x20BB	PMERR_INV_PUTDATA_FORMAT
0x20BC	PMERR_INV_QUERY_ELEMENT_NO
0x20BD	PMERR_INV_RECT
0x20BE	PMERR_INV_REGION_CONTROL
0x20BF	PMERR_INV_REGION_MIX_MODE
0x20C0	PMERR_INV_REPLACE_MODE_FUNC
0x20C1	PMERR_INV_RESERVED_FIELD
0x20C2	PMERR_INV_RESET_OPTIONS
0x20C3	PMERR_INV_RGBCOLOR
0x20C4	PMERR_INV_SCAN_START
0x20C5	PMERR_INV_SEG_ATTR
0x20C6	PMERR_INV_SEG_ATTR_VALUE
0x20C7	PMERR_INV_SEG_CH_LENGTH
0x20C8	PMERR_INV_SEG_NAME
0x20C9	PMERR_INV_SEG_OFFSET
0x20CA	PMERR_INV_SETID
0x20CB	PMERR_INV_SETID_TYPE
0x20CC	PMERR_INV_SET_VIEWPORT_OPTION
0x20CD	PMERR_INV_SHARPNESS_PARM

0x20CE	PMERR_INV_SOURCE_OFFSET
0x20CF	PMERR_INV_STOP_DRAW_VALUE
0x20D0	PMERR_INV_TRANSFORM_TYPE
0x20D1	PMERR_INV_USAGE_PARM
0x20D2	PMERR_INV_VIEWING_LIMITS
0x20D3	PMERR_JFILE_BUSY
0x20D4	PMERR_JNL_FUNC_DATA_TOO_LONG
0x20D5	PMERR_KERNING_NOT_SUPPORTED
0x20D6	PMERR_LABEL_NOT_FOUND
0x20D7	PMERR_MATRIX_OVERFLOW
0x20D8	PMERR_METAFILE_INTERNAL_ERROR
0x20D9	PMERR_METAFILE_IN_USE
0x20DA	PMERR_METAFILE_LIMIT_EXCEEDED
0x20DB	PMERR_NAME_STACK_FULL
0x20DC	PMERR_NOT_CREATED_BY_DEVOPENDC
0x20DD	PMERR_NOT_IN_AREA
0x20DE	PMERR_NOT_IN_DRAW_MODE
0x20DF	PMERR_NOT_IN_ELEMENT
0x20E0	PMERR_NOT_IN_IMAGE
0x20E1	PMERR_NOT_IN_PATH
0x20E2	PMERR_NOT_IN_RETAIN_MODE
0x20E3	PMERR_NOT_IN_SEG
0x20E4	PMERR_NO_BITMAP_SELECTED
0x20E5	PMERR_NO_CURRENT_ELEMENT
0x20E6	PMERR_NO_CURRENT_SEG
0x20E7	PMERR_NO_METAFILE_RECORD_HANDLE
0x20E8	PMERR_ORDER_TOO_BIG
0x20E9	PMERR_OTHER_SET_ID_REFS
0x20EA	PMERR_OVERRAN_SEG
0x20EB	PMERR_OWN_SET_ID_REFS
0x20EC	PMERR_PATH_INCOMPLETE
0x20ED	PMERR_PATH_LIMIT_EXCEEDED
0x20EE	PMERR_PATH_UNKNOWN
0x20EF	PMERR_PEL_IS_CLIPPED
0x20F0	PMERR_PEL_NOT_AVAILABLE
0x20F1	PMERR_PRIMITIVE_STACK_EMPTY
0x20F2	PMERR_PROLOG_ERROR
0x20F3	PMERR_PROLOG_SEG_ATTR_NOT_SET
0x20F4	PMERR_PS_BUSY
0x20F5	PMERR_PS_IS_ASSOCIATED
0x20F6	PMERR_RAM_JNL_FILE_TOO_SMALL
0x20F7	PMERR_REALIZE_NOT_SUPPORTED
0x20F8	PMERR_REGION_IS_CLIP_REGION
0x20F9	PMERR_RESOURCE_DEPLETION
0x20FA	PMERR_SEG_AND_REFSEG_ARE_SAME
0x20FB	PMERR_SEG_CALL_RECURSIVE
0x20FC	PMERR_SEG_CALL_STACK_EMPTY
0x20FD	PMERR_SEG_CALL_STACK_FULL
0x20FE	PMERR_SEG_IS_CURRENT
0x20FF	PMERR_SEG_NOT_CHAINED
0x2100	PMERR_SEG_NOT_FOUND
0x2101	PMERR_SEG_STORE_LIMIT_EXCEEDED
0x2102	PMERR_SETID_IN_USE
0x2103	PMERR_SETID_NOT_FOUND
0x2104	PMERR_STARTDOC_NOT_ISSUED
0x2105	PMERR_STOP_DRAW_OCCURRED
0x2106	PMERR_TOO_MANY_METAFILES_IN_USE
0x2107	PMERR_TRUNCATED_ORDER
0x2108	PMERR_UNCHAINED_SEG_ZERO_INV
0x2109	PMERR_UNSUPPORTED_ATTR
0x210A	PMERR_UNSUPPORTED_ATTR_VALUE
0x210B	PMERR_ENDDOC_NOT_ISSUED
0x210C	PMERR_PS_NOT_ASSOCIATED
0x210D	PMERR_INV_FLOOD_FILL_OPTIONS
0x210E	PMERR_INV_FACENAME
0x210F	PMERR_PALETTE_SELECTED
0x2110	PMERR_NO_PALETTE_SELECTED
0x2111	PMERR_INV_HPAL
0x2112	PMERR_PALETTE_BUSY
0x2113	PMERR_START_POINT_CLIPPED
0x2114	PMERR_NO_FILL
0x2115	PMERR_INV_FACENAMEDESC
0x2116	PMERR_INV_BITMAP_DATA

0x2117	PMERR_INV_CHAR_ALIGN_ATTR
0x2118	PMERR_INV_HFONT
0x2119	PMERR_HFONT_IS_SELECTED
0x2120	PMERR_DRV_R_NOT_SUPPORTED
0x2120	PMERR_RASTER_FONT
0x3001	HMERR_DDF_MEMORY
0x3002	HMERR_DDF_ALIGN_TYPE
0x3003	HMERR_DDF_BACKCOLOR
0x3004	HMERR_DDF_FORECOLOR
0x3005	HMERR_DDF_FONTSTYLE
0x3006	HMERR_DDF_REFTYPE
0x3007	HMERR_DDF_LIST_UNCLOSED
0x3008	HMERR_DDF_LIST_UNINITIALIZED
0x3009	HMERR_DDF_LIST_BREAKTYPE
0x300A	HMERR_DDF_LIST_SPACING
0x300B	HMERR_DDF_HINSTANCE
0x300C	HMERR_DDF_EXCEED_MAX_LENGTH
0x300D	HMERR_DDF_EXCEED_MAX_INC
0x300E	HMERR_DDF_INVALID_DDF
0x300F	HMERR_DDF_FORMAT_TYPE
0x3010	HMERR_DDF_INVALID_PARM
0x3011	HMERR_DDF_INVALID_FONT
0x3012	HMERR_DDF_SEVERE
0x4001	PMERR_SPL_DRIVER_ERROR
0x4001	MERR_SPL_DRIVER_ERROR
0x4002	PMERR_SPL_DEVICE_ERROR
0x4002	MERR_SPL_DEVICE_ERROR
0x4003	PMERR_SPL_DEVICE_NOT_INSTALLED
0x4003	MERR_SPL_DEVICE_NOT_INSTALLED
0x4004	PMERR_SPL_QUEUE_ERROR
0x4004	MERR_SPL_QUEUE_ERROR
0x4005	PMERR_SPL_INV_HSPL
0x4005	MERR_SPL_INV_HSPL
0x4006	PMERR_SPL_NO_DISK_SPACE
0x4006	MERR_SPL_NO_DISK_SPACE
0x4007	PMERR_SPL_NO_MEMORY
0x4007	MERR_SPL_NO_MEMORY
0x4008	PMERR_SPL_PRINT_ABORT
0x4008	MERR_SPL_PRINT_ABORT
0x4009	PMERR_SPL_SPOOLER_NOT_INSTALLED
0x4009	MERR_SPL_SPOOLER_NOT_INSTALLED
0x400A	PMERR_SPL_INV_FORMS_CODE
0x400A	MERR_SPL_INV_FORMS_CODE
0x400B	PMERR_SPL_INV_PRIORITY
0x400B	MERR_SPL_INV_PRIORITY
0x400C	PMERR_SPL_NO_FREE_JOB_ID
0x400C	MERR_SPL_NO_FREE_JOB_ID
0x400D	PMERR_SPL_NO_DATA
0x400D	MERR_SPL_NO_DATA
0x400E	PMERR_SPL_INV_TOKEN
0x400E	MERR_SPL_INV_TOKEN
0x400F	PMERR_SPL_INV_DATATYPE
0x400F	MERR_SPL_INV_DATATYPE
0x4010	PMERR_SPL_PROCESSOR_ERROR
0x4010	MERR_SPL_PROCESSOR_ERROR
0x4011	PMERR_SPL_INV_JOB_ID
0x4011	MERR_SPL_INV_JOB_ID
0x4012	PMERR_SPL_JOB_NOT_PRINTING
0x4012	MERR_SPL_JOB_NOT_PRINTING
0x4013	PMERR_SPL_JOB_PRINTING
0x4013	MERR_SPL_JOB_PRINTING
0x4014	PMERR_SPL_QUEUE_ALREADY_EXISTS
0x4014	MERR_SPL_QUEUE_ALREADY_EXISTS
0x4015	PMERR_SPL_INV_QUEUE_NAME
0x4015	MERR_SPL_INV_QUEUE_NAME
0x4016	PMERR_SPL_QUEUE_NOT_EMPTY
0x4016	MERR_SPL_QUEUE_NOT_EMPTY
0x4017	PMERR_SPL_DEVICE_ALREADY_EXISTS
0x4017	MERR_SPL_DEVICE_ALREADY_EXISTS
0x4018	PMERR_SPL_DEVICE_LIMIT_REACHED
0x4018	MERR_SPL_DEVICE_LIMIT_REACHED
0x4019	PMERR_SPL_STATUS_STRING_TRUNC
0x4019	MERR_SPL_STATUS_STRING_TRUNC

0x401A	PMERR_SPL_INV_LENGTH_OR_COUNT
0x401A	MERR_SPL_INV_LENGTH_OR_COUNT
0x401B	PMERR_SPL_FILE_NOT_FOUND
0x401B	MERR_SPL_FILE_NOT_FOUND
0x401C	PMERR_SPL_CANNOT_OPEN_FILE
0x401C	MERR_SPL_CANNOT_OPEN_FILE
0x401D	PMERR_SPL_DRIVER_NOT_INSTALLED
0x401D	MERR_SPL_DRIVER_NOT_INSTALLED
0x401E	PMERR_SPL_INV_PROCESSOR_DATATYPE
0x401E	MERR_SPL_INV_PROCESSOR_DATATYPE
0x401F	PMERR_SPL_INV_DRIVER_DATATYPE
0x401F	MERR_SPL_INV_DRIVER_DATATYPE
0x4020	PMERR_SPL_PROCESSOR_NOT_INST
0x4020	MERR_SPL_PROCESSOR_NOT_INST
0x4021	PMERR_SPL_NO_SUCH_LOG_ADDRESS
0x4021	MERR_SPL_NO_SUCH_LOG_ADDRESS
0x4022	PMERR_SPL_PRINTER_NOT_FOUND
0x4022	MERR_SPL_PRINTER_NOT_FOUND
0x4023	PMERR_SPL_DD_NOT_FOUND
0x4023	MERR_SPL_DD_NOT_FOUND
0x4024	PMERR_SPL_QUEUE_NOT_FOUND
0x4024	MERR_SPL_QUEUE_NOT_FOUND
0x4025	PMERR_SPL_MANY_QUEUES_ASSOC
0x4025	MERR_SPL_MANY_QUEUES_ASSOC
0x4026	PMERR_SPL_NO_QUEUES_ASSOCIATED
0x4026	MERR_SPL_NO_QUEUES_ASSOCIATED
0x4027	PMERR_SPL_INI_FILE_ERROR
0x4027	MERR_SPL_INI_FILE_ERROR
0x4028	PMERR_SPL_NO_DEFAULT_QUEUE
0x4028	MERR_SPL_NO_DEFAULT_QUEUE
0x4029	PMERR_SPL_NO_CURRENT_FORMS_CODE
0x4029	MERR_SPL_NO_CURRENT_FORMS_CODE
0x402A	PMERR_SPL_NOT_AUTHORISED
0x402A	MERR_SPL_NOT_AUTHORISED
0x402B	PMERR_SPL_TEMP_NETWORK_ERROR
0x402B	MERR_SPL_TEMP_NETWORK_ERROR
0x402C	PMERR_SPL_HARD_NETWORK_ERROR
0x402C	MERR_SPL_HARD_NETWORK_ERROR
0x402D	PMERR_DEL_NOT_ALLOWED
0x402D	MERR_DEL_NOT_ALLOWED
0x402E	PMERR_CANNOT_DEL_QP_REF
0x402E	MERR_CANNOT_DEL_QP_REF
0x402F	PMERR_CANNOT_DEL_QNAME_REF
0x402F	MERR_CANNOT_DEL_QNAME_REF
0x4030	PMERR_CANNOT_DEL_PRINTER_DD_REF
0x4030	MERR_CANNOT_DEL_PRINTER_DD_REF
0x4031	PMERR_CANNOT_DEL_PRN_NAME_REF
0x4031	MERR_CANNOT_DEL_PRN_NAME_REF
0x4032	PMERR_CANNOT_DEL_PRN_ADDR_REF
0x4032	MERR_CANNOT_DEL_PRN_ADDR_REF
0x4033	PMERR_SPOOLER_QP_NOT_DEFINED
0x4033	MERR_SPOOLER_QP_NOT_DEFINED
0x4034	PMERR_PRN_NAME_NOT_DEFINED
0x4034	MERR_PRN_NAME_NOT_DEFINED
0x4035	PMERR_PRN_ADDR_NOT_DEFINED
0x4035	MERR_PRN_ADDR_NOT_DEFINED
0x4036	PMERR_PRINTER_DD_NOT_DEFINED
0x4036	MERR_PRINTER_DD_NOT_DEFINED
0x4037	PMERR_PRINTER_QUEUE_NOT_DEFINED
0x4037	MERR_PRINTER_QUEUE_NOT_DEFINED
0x4038	PMERR_PRN_ADDR_IN_USE
0x4038	MERR_PRN_ADDR_IN_USE
0x4039	PMERR_SPL_TOO_MANY_OPEN_FILES
0x4039	MERR_SPL_TOO_MANY_OPEN_FILES
0x403A	PMERR_SPL_CP_NOT_REQD
0x403A	MERR_SPL_CP_NOT_REQD
0x4040	PMERR_UNABLE_TO_CLOSE_DEVICE
0x4040	MERR_UNABLE_TO_CLOSE_DEVICE
0x4FC9	PMERR_SPLMSGBOX_INFO_CAPTION
0x4FC9	MERR_SPLMSGBOX_INFO_CAPTION
0x4FCA	PMERR_SPLMSGBOX_WARNING_CAPTION
0x4FCA	MERR_SPLMSGBOX_WARNING_CAPTION
0x4FCB	PMERR_SPLMSGBOX_ERROR_CAPTION

0x4FCB	MERR_SPLMSGBOX_ERROR_CAPTION
0x4FCC	PMERR_SPLMSGBOX_SEVERE_CAPTION
0x4FCC	MERR_SPLMSGBOX_SEVERE_CAPTION
0x4FCD	PMERR_SPLMSGBOX_JOB_DETAILS
0x4FCD	MERR_SPLMSGBOX_JOB_DETAILS
0x4FCE	PMERR_SPLMSGBOX_ERROR_ACTION
0x4FCE	MERR_SPLMSGBOX_ERROR_ACTION
0x4FCF	PMERR_SPLMSGBOX_SEVERE_ACTION
0x4FCF	MERR_SPLMSGBOX_SEVERE_ACTION
0x4FD0	PMERR_SPLMSGBOX_BIT_0_TEXT
0x4FD0	MERR_SPLMSGBOX_BIT_0_TEXT
0x4FD1	PMERR_SPLMSGBOX_BIT_1_TEXT
0x4FD1	MERR_SPLMSGBOX_BIT_1_TEXT
0x4FD2	PMERR_SPLMSGBOX_BIT_2_TEXT
0x4FD2	MERR_SPLMSGBOX_BIT_2_TEXT
0x4FD3	PMERR_SPLMSGBOX_BIT_3_TEXT
0x4FD3	MERR_SPLMSGBOX_BIT_3_TEXT
0x4FD4	PMERR_SPLMSGBOX_BIT_4_TEXT
0x4FD4	MERR_SPLMSGBOX_BIT_4_TEXT
0x4FD5	PMERR_SPLMSGBOX_BIT_5_TEXT
0x4FD5	MERR_SPLMSGBOX_BIT_5_TEXT
0x4FD6	PMERR_SPLMSGBOX_BIT_15_TEXT
0x4FD6	MERR_SPLMSGBOX_BIT_15_TEXT
0x4FD7	PMERR_SPL_NOPATHBUFFER
0x4FD7	MERR_SPL_NOPATHBUFFER
0x4FD8	MERR_SPL_ALREADY_INITIALISED
0x4FD9	MERR_SPL_ERROR
0x5001	PMERR_INV_TYPE
0x5001	MERR_INV_TYPE
0x5002	PMERR_INV_CONV
0x5002	MERR_INV_CONV
0x5003	PMERR_INV_SEGLEN
0x5003	MERR_INV_SEGLEN
0x5004	PMERR_DUP_SEGNAME
0x5004	MERR_DUP_SEGNAME
0x5005	PMERR_INV_XFORM
0x5005	MERR_INV_XFORM
0x5006	PMERR_INV_VIEWLIM
0x5006	MERR_INV_VIEWLIM
0x5007	PMERR_INV_3DCOORD
0x5007	MERR_INV_3DCOORD
0x5008	PMERR_SMB_OVFLOW
0x5008	MERR_SMB_OVFLOW
0x5009	PMERR_SEG_OVFLOW
0x5009	MERR_SEG_OVFLOW
0x5010	PMERR_PIC_DUP_FILENAME
0x5010	MERR_PIC_DUP_FILENAME
SPLERR_BASE+0FA1	PMERR_SPL_ERROR_1
SPLERR_BASE+0FA2	PMERR_SPL_ERROR_2
SPLERR_BASE+0FA3	PMERR_SPL_ERROR_3
SPLERR_BASE+0FA4	PMERR_SPL_ERROR_4
SPLERR_BASE+0FA5	PMERR_SPL_ERROR_5
SPLERR_BASE+0FA6	PMERR_SPL_ERROR_6
SPLERR_BASE+0FA7	PMERR_SPL_ERROR_7
SPLERR_BASE+0FA8	PMERR_SPL_ERROR_8
SPLERR_BASE+0FA9	PMERR_SPL_ERROR_9
SPLERR_BASE+0FAA	PMERR_SPL_ERROR_10
SPLERR_BASE+0FAB	PMERR_SPL_ERROR_11
SPLERR_BASE+0FAC	PMERR_SPL_ERROR_12
SPLERR_BASE+0FAD	PMERR_SPL_ERROR_13
SPLERR_BASE+0FAE	PMERR_SPL_ERROR_14
SPLERR_BASE+0FAF	PMERR_SPL_ERROR_15
SPLERR_BASE+0FB0	PMERR_SPL_ERROR_16
SPLERR_BASE+0FB1	PMERR_SPL_ERROR_17
SPLERR_BASE+0FB2	PMERR_SPL_ERROR_18
SPLERR_BASE+0FB3	PMERR_SPL_ERROR_19
SPLERR_BASE+0FB4	PMERR_SPL_ERROR_20
SPLERR_BASE+0FB5	PMERR_SPL_ERROR_21
SPLERR_BASE+0FB6	PMERR_SPL_ERROR_22
SPLERR_BASE+0FB7	PMERR_SPL_ERROR_23
SPLERR_BASE+0FB8	PMERR_SPL_ERROR_24
SPLERR_BASE+0FB9	PMERR_SPL_ERROR_25
SPLERR_BASE+0FBA	PMERR_SPL_ERROR_26

SPLERR_BASE+0FBB	PMERR_SPL_ERROR_27
SPLERR_BASE+0FBC	PMERR_SPL_ERROR_28
SPLERR_BASE+0FBD	PMERR_SPL_ERROR_29
SPLERR_BASE+0FBE	PMERR_SPL_ERROR_30
SPLERR_BASE+0FBF	PMERR_SPL_ERROR_31
SPLERR_BASE+0FC0	PMERR_SPL_ERROR_32
SPLERR_BASE+0FC1	PMERR_SPL_ERROR_33
SPLERR_BASE+0FC2	PMERR_SPL_ERROR_34
SPLERR_BASE+0FC3	PMERR_SPL_ERROR_35
SPLERR_BASE+0FC4	PMERR_SPL_ERROR_36
SPLERR_BASE+0FC5	PMERR_SPL_ERROR_37
SPLERR_BASE+0FC6	PMERR_SPL_ERROR_38
SPLERR_BASE+0FC7	PMERR_SPL_ERROR_39
SPLERR_BASE+0FC8	PMERR_SPL_ERROR_40
SPLERR_BASE+0FFF	PMERR_SPL_ERROR
SPLERR_BASE+0FFD	PMERR_SPL_ALREADY_INITIALISED

Error Name and Explanation

This appendix gives an explanation for each PM error. The errors are listed in alphabetic order. The number associated with each error is given in [Error Number and Name](#).

Error Constant	Explanation
HMERR_ALLOCATE_SEGMENT	Unable to allocate a segment of memory for memory allocation requests from the Help Manager.
HMERR_CLOSE_LIB_FILE	The library file cannot be closed.
HMERR_CONTENT_NOT_FOUND	The library file does not have any content.
HMERR_DATABASE_NOT_OPEN	Unable to read the unopened database.
HMERR_DDF_ALIGN_TYPE	The alignment type is not valid.
HMERR_DDF_BACKCOLOR	The background color is not valid.
HMERR_DDF_EXCEED_MAX_INC	The value specified to increment DDF memory is too large.
HMERR_DDF_EXCEED_MAX_LENGTH	The amount of data is too large for the DDF buffer.
HMERR_DDF_FONTSTYLE	The font style is not valid.
HMERR_DDF_FORECOLOR	The foreground color is not valid.
HMERR_DDF_FORMAT_TYPE	The format type specified is invalid.
HMERR_DDF_HINSTANCE	The DDF instance is invalid.
HMERR_DDF_INVALID_DDF	The DDF handle is invalid.
HMERR_DDF_INVALID_FONT	The font value specified is invalid.

HMERR_DDF_INVALID_PARM	One of the DDF parameters specified is invalid.
HMERR_DDF_LIST_BREAKTYPE	The value of BreakType is not valid.
HMERR_DDF_LIST_SPACING	The value for Spacing is not valid.
HMERR_DDF_LIST_UNCLOSED	An attempt was made to nest a list.
HMERR_DDF_LIST_UNINITIALIZED	No definition list has been initialized by DdfBeginList .
HMERR_DDF_MEMORY	Not enough memory is available.
HMERR_DDF_REFTYPE	The reference type is not valid.
HMERR_DDF_SEVERE	Internal error detected by the Help Manager.
HMERR_FREE_MEMORY	Unable to free allocated memory.
HMERR_HELP_INST_CALLED_INVALID	The handle of the instance specified on a call to the Help Manager does not have the class name of a Help Manager instance.
HMERR_HELP_INSTANCE_UNDEFINE	The help instance handle specified is invalid.
HMERR_HELPITEM_NOT_FOUND	Context-sensitive help was requested but the ID of the main help item specified was not found in the help table.
HMERR_HELPSUBITEM_NOT_FOUND	Context-sensitive help was requested but the ID of the help item specified was not found in the help subtable.
HMERR_HELPTABLE_UNDEFINE	The application did not provide a help table for context-sensitive help.
HMERR_INDEX_NOT_FOUND	The index is not in the library file.
HMERR_INVALID_ASSOC_APP_WND	The application window handle specified on the WinAssociateHelpInstance function is not a valid window handle.
HMERR_INVALID_ASSOC_HELP_INST	The help instance handle specified on the WinAssociateHelpInstance function is not a valid window handle.
HMERR_INVALID_DESTROY_HELP_INST	The window handle specified as the help instance to destroy is not of the help instance class.
HMERR_INVALID_HELP_INSTANCE_HDL	The handle specified to be a help instance does not have the class name of a Help

	Manager instance.
HMERR_INVALID_HELPSUBITEM_SIZE	The help subtable item size is less than 2.
HMERR_INVALID_LIB_FILE	Improper library file provided.
HMERR_INVALID_QUERY_APP_WND	The application window specified on a WinQueryHelpInstance function is not a valid window handle.
HMERR_LOAD_DLL	Unable to load resource data link library.
HMERR_NO_FRAME_WND_IN_CHAIN	There is no frame window in the window chain from which to find or set the associated help instance.
HMERR_NO_HELP_INST_IN_CHAIN	The parent or owner chain of the application window specified does not have an associated help instance.
HMERR_NO_MEMORY	Unable to allocate the requested amount of memory.
HMERR_OPEN_LIB_FILE	The library file cannot be opened.
HMERR_PANEL_NOT_FOUND	Unable to find the requested help panel.
HMERR_READ_LIB_FILE	The library file cannot be read.
PMERR_ACCESS_DENIED	The memory block was not allocated properly.
PMERR_ALREADY_IN_AREA	An attempt was made to begin a new area while an existing area bracket was already open.
PMERR_ALREADY_IN_ELEMENT	An attempt was made to begin a new element while an existing element bracket was already open.
PMERR_ALREADY_IN_PATH	An attempt was made to begin a new path while an existing path bracket was already open.
PMERR_ALREADY_IN_SEG	An attempt was made to open a new segment while an existing segment bracket was already open.
PMERR_APPL_STRUCTURE_TOO_SMALL	The application buffer length is less than the total length required for the (application) component types.
PMERR_AREA_INCOMPLETE	One of the following has occurred: <ul style="list-style-type: none"> o A segment has been opened , closed, or drawn. o GpiAssociate was issued while an area bracket was open.

	o A drawn segment has opened an area with a bracket and ended without closing it.
PMERR_ARRAY_TOO_LARGE	More than 4 bytes was attempted to be inserted or extracted.
PMERR_ARRAY_TOO_SMALL	The array specified was too small.
PMERR_ATOM_NAME_NOT_FOUND	The specified atom name is not in the atom table.
PMERR_BASE_ERROR	An OS/2 base error has occurred. The base error code can be accessed using the OffBinaryData field of the ERRINFO structure returned by WinGetErrorInfo .
PMERR_BITMAP_IN_USE	An attempt was made either to set a bit map into a device context using GpiSetBitmap while it was already selected into an existing device context, or to tag a bit map with a local pattern set identifier (setid) using GpiSetBitmapId while it was already tagged with an existing setid.
PMERR_BITMAP_IS_SELECTED	An attempt was made to delete a bit map while it was selected into a device context.
PMERR_BITMAP_NOT_FOUND	A attempt was made to perform a bit-map operation on a bit map that did not exist.
PMERR_BITMAP_NOT_SELECTED	A attempt was made to perform an operation on presentation space associated with a memory device context that had no selected bit map.
PMERR_BOUNDS_OVERFLOW	An internal overflow error occurred during boundary data accumulation. This can occur if coordinates or matrix transformation elements (or both) are invalid or too large.
PMERR_BUFFER_TOO_SMALL	The supplied buffer was not large enough for the data to be returned.
PMERR_C_LENGTH_TOO_SMALL	The maximum length of the C structure is less than the total length required for the (C) component types.
PMERR_CALLED_SEG_IS_CHAINED	An attempt was made to call a segment that has a chained attribute set.
PMERR_CALLED_SEG_IS_CURRENT	An attempt was made to call a segment that is currently open.

PMERR_CALLED_SEG_NOT_FOUND	An attempt was made to call a segment that did not exist.
PMERR_CAN_NOT_CALL_SPOOLER	An error occurred attempting to call the spooler validation routine. This error is not raised if the spooler is not installed.
PMERR_CANNOT_DEL_PRINTER_DD_REF	Presentation Manager device driver deletion not possible due to a reference.
PMERR_CANNOT_DEL_PRN_ADDR_REF	Printer port deletion not possible due to a reference.
PMERR_CANNOT_DEL_PRN_NAME_REF	Printer deletion not possible due to a reference.
PMERR_CANNOT_DEL_QNAME_REF	Spooler queue deletion not possible due to a reference.
PMERR_CANNOT_DEL_QP_REF	Spooler queue processor deletion not possible due to a reference.
PMERR_CANNOT_SET_FOCUS	Focus cannot be set if a focus change is in progress, or if a system-modal window exists.
PMERR_CANNOT_STOP	The session cannot be stopped.
PMERR_COL_TABLE_NOT_REALIZABLE	An attempt was made to realize a color table that is not realizable.
PMERR_COL_TABLE_NOT_REALIZED	An attempt was made to realize a color table on a device driver that does not support this function.
PMERR_COORDINATE_OVERFLOW	An internal coordinate overflow error occurred. This can occur if coordinates or matrix transformation elements (or both) are invalid or too large.
PMERR_DATA_TOO_LONG	An attempt was made to transfer more than the maximum permitted amount of data (64512 bytes) using GpiPutData, GpiGetData, or GpiElement.
PMERR_DATATYPE_ENTRY_BAD_INDEX	An invalid datatype entry index was specified.
PMERR_DATATYPE_ENTRY_CTL_BAD	An invalid datatype entry control was specified.
PMERR_DATATYPE_ENTRY_CTL_MISS	The datatype entry control was missing.
PMERR_DATATYPE_ENTRY_NOT_NUM	The datatype entry specified was not numerical.
PMERR_DATATYPE_ENTRY_NOT_OFF	The datatype entry specified was not an offset.
PMERR_DATATYPE_INVALID	An invalid datatype was

PMERR_DATATYPE_NOT_UNIQUE	specified. An attempt to register a datatype failed because it is not unique.
PMERR_DATATYPE_TOO_LONG	The datatype specified was too long.
PMERR_DATATYPE_TOO_SMALL	The datatype specified was too small.
PMERR_DC_IS_ASSOCIATED	An attempt was made to associate a presentation space with a device context that was already associated or to destroy a device context that was associated.
PMERR_DEL_NOT_ALLOWED	Deletion not possible.
PMERR_DESC_STRING_TRUNCATED	An attempt was made to supply a description string with GpiBeginElement that was greater than the permitted maximum length (251 characters). The string was truncated.
PMERR_DEV_FUNC_NOT_INSTALLED	The function requested is not supported by the presentation driver.
PMERR_DEVICE_DRIVER_ERROR_1	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_10	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_2	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_3	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_4	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_5	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_6	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_7	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_8	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_9	Miscellaneous error available for use by user written device drivers.

PMERR_DOS_ERROR	A DOS call returned an error.
PMERR_DOSOPEN_FAILURE	A DosOpen call made during GpiLoadMetaFile or GpiSaveMetaFile gave a good return code but the file was not opened successfully.
PMERR_DOSREAD_FAILURE	A DosRead call made during GpiLoadMetaFile gave a good return code. However, it failed to read any more bytes although the file length indicated that there were more to be read.
PMERR_DRIVER_NOT_FOUND	The device driver specified with DevPostDeviceModes was not found.
PMERR_DUP_SEG	During GpiPlayMetaFile, while the actual drawing mode was draw-and-retain or retain , a metafile segment to be stored in the presentation space was found to have the same segment identifier as an existing segment.
PMERR_DUP_SEGNAME	A called segment has a name that has already been used by another called segment in the input PIF.
PMERR_DUPLICATE_TITLE	The program title specified in the PIBSTRUCT already exists within the same group.
PMERR_DYNAMIC_SEG_SEQ_ERROR	During removal of dynamic segments while processing GpiDrawChain, GpiDrawFrom, or GpiDrawSegment, the internal state indicated that dynamic segment data was still visible after all chained dynamic segments had been processed. This can occur if segments drawn dynamically (including called segments) are modified or removed from the chain while visible.
PMERR_DYNAMIC_SEG_ZERO_INV	An attempt was been made to open a dynamic segment with a segment identifier of zero.
PMERR_ENDDOC_NOT_ISSUED	A request to close the spooled output without first issuing a an ENDDOC was attempted.
PMERR_ESC_CODE_NOT_SUPPORTED	The code specified with DevEscape is not supported by the target device driver.
PMERR_EXCEEDS_MAX_SEG_LENGTH	During metafile creation or generation of retained graphics the system has exceeded maximum segment size.

PMERR_FONT_AND_MODE_MISMATCH	An attempt was made to draw characters with a character mode and character set that are incompatible. For example, the character specifies an image/raster font when the mode calls for a vector/outline font.
PMERR_FONT_FILE_NOT_LOADED	An attempt was made to unload a font file that was not loaded.
PMERR_FONT_NOT_LOADED	An attempt was made to create a font that was not loaded.
PMERR_FUNCTION_NOT_SUPPORTED	The function is not supported.
PMERR_GREATER_THAN_64K	A data item or array dimension is greater than 65 535.
PMERR_HBITMAP_BUSY	An internal bit map busy error was detected. The bit map was locked by one thread during an attempt to access it from another thread.
PMERR_HDC_BUSY	An internal device context busy error was detected. The device context was locked by one thread during an attempt to access it from another thread.
PMERR_HEAP_MAX_SIZE_REACHED	The heap has reached its maximum size (64KB), and cannot be increased.
PMERR_HEAP_OUT_OF_MEMORY	An attempt to increase the size of the heap failed.
PMERR_HFONT_IS_SELECTED	An attempt has been made to either change the owner of a font, or delete when it is currently selected.
PMERR_HRGN_BUSY	An internal region busy error was detected. The region was locked by one thread during an attempt to access it from another thread.
PMERR_HUGE_FONTS_NOT_SUPPORTED	An attempt was made using GpiSetCharSet, GpiSetPatternSet, GpiSetMarkerSet, or GpiSetAttrs to select a font that is larger than the maximum size (64Kb) supported by the target device driver.
PMERR_ID_HAS_NO_BITMAP	No bit map was tagged with the setid specified on a GpiQueryBitmapHandle function.
PMERR_IMAGE_INCOMPLETE	A drawn segment has

	opened an image bracket and ended without closing it.
PMERR_INCOMPATIBLE_BITMAP	An attempt was made to select a bit map or perform a BitBlt operation on a device context that was incompatible with the format of the bit map.
PMERR_INCOMPATIBLE_METAFILE	An attempt was made to associate a presentation space and a metafile device context with incompatible page units, size or coordinate format; or to play a metafile using the RES_RESET option (to reset the presentation space) to a presentation space that is itself associated with a metafile device context.
PMERR_INCORRECT_DATATYPE	A data type is specified which is incorrect for this function.
PMERR_INCORRECT_DC_TYPE	An attempt was made to perform a bit-map operation on a presentation space associated with a device context of a type that is unable to support bit-map operations.
PMERR_INCORRECT_HSTRUCT	A structure handle is non-NULL, and is invalid for one of the following reasons: <ul style="list-style-type: none"> o It is not the handle of a data structure. o It is the handle of an ERRINFO structure, which should not be used in this call. o A handle block returned by the binding to the application has been used for an in-line structure handle.
PMERR_INI_FILE_IS_SYS_OR_USER	User or system initialization file cannot be closed.
PMERR_INSUFF_SPACE_TO_ADD	The initialization file could not be extended to add the required program or group.
PMERR_INSUFFICIENT_DISK_SPACE	The operation terminated through insufficient disk space.
PMERR_INSUFFICIENT_MEMORY	The operation terminated through insufficient memory.
PMERR_INV_3DCOORD	An order specifying 3-dimensional coordinates has been found in the input PIF.
PMERR_INV_ANGLE_PARM	An invalid angle parameter was specified with GpiPartialArc.
PMERR_INV_ARC_CONTROL	An invalid control parameter was specified with GpiFullArc.
PMERR_INV_AREA_CONTROL	An invalid options parameter

	was specified with GpiBeginArea.
PMERR_INV_ATTR_MODE	An invalid mode parameter was specified with GpiSetAttrMode.
PMERR_INV_BACKGROUND_COL_ATTR	An invalid background color attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_BACKGROUND_MIX_ATTR	An invalid background mix attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_BITBLT_MIX	An invalid <i>/Rop</i> was specified with a GpiBitBlt or GpiWCBitBlt function.
PMERR_INV_BITBLT_STYLE	An invalid options parameter was specified with a GpiBitBlt or GpiWCBitBlt function.
PMERR_INV_BITMAP_DATA	In processing a bit map, the end of the data was unexpectedly encountered.
PMERR_INV_BITMAP_DIMENSION	An invalid dimension was specified with a load bit-map function.
PMERR_INV_BOX_CONTROL	An invalid control parameter was specified with GpiBox.
PMERR_INV_BOX_ROUNDING_PARM	An invalid corner rounding control parameter was specified with GpiBox.
PMERR_INV_CHAR_ALIGN_ATTR	The text alignment attribute specified in GpiSetTextAlignment is not valid.
PMERR_INV_CHAR_ANGLE_ATTR	The default character angle attribute value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_CHAR_DIRECTION_ATTR	An invalid character direction attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_CHAR_MODE_ATTR	An invalid character mode attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_CHAR_POS_OPTIONS	An invalid options parameter was specified with GpiCharStringPos or GpiCharStringPosAt.

PMERR_INV_CHAR_SET_ATTR	An invalid character setid attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_CHAR_SHEAR_ATTR	An invalid character shear attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_CLIP_PATH_OPTIONS	An invalid options parameter was specified with GpiSetClipPath.
PMERR_INV_CODEPAGE	An invalid code-page was specified.
PMERR_INV_COLOR_ATTR	An invalid color attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_COLOR_DATA	Invalid color table definition data was specified with GpiCreateLogColorTable.
PMERR_INV_COLOR_FORMAT	An invalid format parameter was specified with GpiCreateLogColorTable.
PMERR_INV_COLOR_INDEX	An invalid color index parameter was specified with GpiQueryRGBColor.
PMERR_INV_COLOR_OPTIONS	An invalid options parameter was specified with a logical color table or color query function.
PMERR_INV_COLOR_START_INDEX	An invalid starting index parameter was specified with a logical color table or color query function.
PMERR_INV_CONV	Invalid conversion-type parameter.
PMERR_INV_COORD_OFFSET	An invalid coordinate offset value was specified.
PMERR_INV_COORD_SPACE	An invalid source or target coordinate space parameter was specified with GpiConvert.
PMERR_INV_COORDINATE	An invalid coordinate value was specified.
PMERR_INV_CORRELATE_DEPTH	An invalid maxdepth parameter was specified with GpiCorrelateSegment, GpiCorrelateFrom, or GpiCorrelateChain.
PMERR_INV_CORRELATE_TYPE	An invalid type parameter was specified with

	GpiCorrelateSegment, GpiCorrelateFrom, or GpiCorrelateChain.
PMERR_INV_CURSOR_BITMAP	An invalid pointer was referenced with WinSetPointer .
PMERR_INV_DC_DATA	An invalid data parameter was specified with DevOpenDC .
PMERR_INV_DC_TYPE	An invalid type parameter was specified with DevOpenDC , or a function was issued that is invalid for a OD_METAFILE_NOQUERY device context.
PMERR_INV_DEV_MODES_OPTIONS	An invalid options parameter was specified with DevPostDeviceModes .
PMERR_INV_DEVICE_NAME	An invalid devicename parameter was specified with DevPostDeviceModes .
PMERR_INV_DRAW_BORDER_OPTION	An invalid option parameter was specified with WinDrawBorder .
PMERR_INV_DRAW_CONTROL	An invalid control parameter was specified with GpiSetDrawControl or GpiQueryDrawControl.
PMERR_INV_DRAW_VALUE	An invalid value parameter was specified with GpiSetDrawControl.
PMERR_INV_DRAWING_MODE	An invalid mode parameter was specified with GpiSetDrawControl not draw-and-retain or draw .
PMERR_INV_DRIVER_DATA	Invalid driver data was specified.
PMERR_INV_DRIVER_NAME	A driver name was specified which has not been installed.
PMERR_INV_EDIT_MODE	An invalid mode parameter was specified with GpiSetEditMode.
PMERR_INV_ELEMENT_OFFSET	An invalid off (offset) parameter was specified with GpiQueryElement.
PMERR_INV_ELEMENT_POINTER	An attempt was made to issue GpiPutData with the element pointer not pointing at the last element.
PMERR_INV_END_PATH_OPTIONS	An attempt to create or delete a path out of context of the path bracket was made.
PMERR_INV_ESC_CODE	An invalid escape code was used in a call to DevEscape .

PMERR_INV_ESCAPE_DATA	An invalid data parameter was specified with DevEscape .
PMERR_INV_FACENAME	An invalid font family name was passed to GpiQueryFaceString.
PMERR_INV_FACENAMEDESC	The font facename description is invalid.
PMERR_INV_FILL_PATH_OPTIONS	An invalid options parameter was specified with GpiFillPath.
PMERR_INV_FIRST_CHAR	An invalid firstchar parameter was specified with GpiQueryWidthTable.
PMERR_INV_FLOOD_FILL_OPTIONS	Invalid flood fill parameters were specified.
PMERR_INV_FONT_ATTRS	An invalid attrs parameter was specified with GpiCreateLogFont.
PMERR_INV_FONT_FILE_DATA	The font file specified with GpiLoadFonts, GpiLoadPublicFonts, GpiQueryFontFileDescriptions, or GpiQueryFullFontFileDescs contains invalid data.
PMERR_INV_FOR_THIS_DC_TYPE	An attempt has been made to issue GpiRemoveDynamics or GpiDrawDynamics to a presentation space associated with a metafile device context.
PMERR_INV_FORMS_CODE	An invalid forms code parameter was specified with DevQueryHardcopyCaps .
PMERR_INV_GEOM_LINE_WIDTH_ATTR	An invalid geometric line width attribute value was specified.
PMERR_INV_GETDATA_CONTROL	An invalid format parameter was specified with GpiGetData.
PMERR_INV_GRAPHICS_FIELD	An invalid field parameter was specified with GpiSetGraphicsField.
PMERR_INV_HBITMAP	An invalid bit-map handle was specified.
PMERR_INV_HDC	An invalid device-context handle or (micro presentation space) presentation-space handle was specified.
PMERR_INV_HFONT	An invalid font handle was specified.
PMERR_INV_HMF	An invalid metafile handle was specified.
PMERR_INV_HPAL	An invalid color palette handle was specified.

PMERR_INV_HPS	An invalid presentation-space handle was specified.
PMERR_INV_HRGN	An invalid region handle was specified.
PMERR_INV_ID	An invalid <i>IPSid</i> parameter was specified with GpiRestorePS.
PMERR_INV_IMAGE_DATA_LENGTH	An invalid <i>Length</i> parameter was specified with GpImage. There is a mismatch between the image size and the data length.
PMERR_INV_IMAGE_DIMENSION	An invalid <i>psizImageSize</i> parameter was specified with GpImage.
PMERR_INV_IMAGE_FORMAT	An invalid <i>Format</i> parameter was specified with GpImage.
PMERR_INV_IN_AREA	An attempt was made to issue a function invalid inside an area bracket. This can be detected while the actual drawing mode is draw or draw-and-retain or during segment drawing or correlation functions.
PMERR_INV_IN_CURRENT_EDIT_MODE	An attempt was made to issue a function invalid inside the current editing mode.
PMERR_INV_IN_ELEMENT	An attempt was made to issue a function invalid inside an element bracket.
PMERR_INV_IN_IMAGE	An attempt was made to issue a function invalid inside an element bracket.
PMERR_INV_IN_PATH	An attempt was made to issue a function invalid inside a path bracket.
PMERR_INV_IN_RETAIN_MODE	An attempt was made to issue a function (for example, query) that is invalid when the actual drawing mode is not draw or draw-and-retain .
PMERR_INV_IN_SEG	An attempt was made to issue a function invalid inside a segment bracket.
PMERR_INV_IN_VECTOR_SYMBOL	An invalid order was detected inside a vector symbol definition while drawing a vector (outline) font.
PMERR_INV_INFO_TABLE	An invalid bit-map info table was specified with a bit-map operation.
PMERR_INV_LENGTH_OR_COUNT	An invalid length or count parameter was specified.
PMERR_INV_LINE_END_ATTR	An invalid line end attribute

PMERR_INV_LINE_JOIN_ATTR	value was specified. An invalid line join attribute value was specified.
PMERR_INV_LINE_TYPE_ATTR	An invalid line type attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_LINE_WIDTH_ATTR	An invalid line width attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_LOGICAL_ADDRESS	An invalid device logical address was specified.
PMERR_INV_MARKER_BOX_ATTR	An invalid marker box attribute value was specified.
PMERR_INV_MARKER_SET_ATTR	An invalid marker set attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_MARKER_SYMBOL_ATTR	An invalid marker symbol attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_MATRIX_ELEMENT	An invalid transformation matrix element was specified.
PMERR_INV_MAX_HITS	An invalid maxhits parameter was specified with GpiCorrelateSegment, GpiCorrelateFrom, or GpiCorrelateChain.
PMERR_INV_METAFILE	An invalid metafile was specified with GpiPlayMetaFile.
PMERR_INV_METAFILE_LENGTH	An invalid length parameter was specified with GpiSetMetaFileBits or GpiQueryMetaFileBits.
PMERR_INV_METAFILE_OFFSET	An invalid length parameter was specified with GpiSetMetaFileBits or GpiQueryMetaFileBits.
PMERR_INV_MICROPS_DRAW_CONTROL	A draw control parameter was specified with GpiSetDrawControl that is invalid in a micro presentation space.
PMERR_INV_MICROPS_FUNCTION	An attempt was made to issue a function that is invalid in a micro presentation space.

PMERR_INV_MICROPS_ORDER	An attempt was made to play a metafile containing orders that are invalid in a micro presentation space.
PMERR_INV_MIX_ATTR	An invalid mix attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_MODE_FOR_OPEN_DYN	An attempt was made to open a segment with the ATTR_DYNAMIC segment set, while the drawing mode was set to DM_DRAW or DM_DRAWANDRETAIN.
PMERR_INV_MODE_FOR_REOPEN_SEG	An attempt was made to reopen an existing segment while the drawing mode was set to DM_DRAW or DM_DRAWANDRETAIN.
PMERR_INV_MODIFY_PATH_MODE	An invalid mode parameter was specified with GpiModifyPath.
PMERR_INV_MULTIPLIER	An invalid multiplier parameter was specified with GpiPartialArc or GpiFullArc.
PMERR_INV_NESTED_FIGURES	Nested figures have been detected within a path definition.
PMERR_INV_OR_INCOMPAT_OPTIONS	An invalid or incompatible (with micro presentation space) options parameter was specified with GpiCreatePS or GpiSetPS.
PMERR_INV_ORDER_LENGTH	An invalid order length was detected during GpiPutData or segment drawing.
PMERR_INV_ORDERING_PARM	An invalid order parameter was specified with GpiSetSegmentPriority.
PMERR_INV_OUTSIDE_DRAW_MODE	An attempt was made to issue a GpiSavePS or GpiRestorePS function, or an output only function (for example, GpiPaintRegion) from GpiPlayMetaFile without the drawing mode set to DM_DRAW.
PMERR_INV_PAGE_VIEWPORT	An invalid viewport parameter was specified with GpiSetPageViewport.
PMERR_INV_PATH_ID	An invalid path identifier parameter was specified.
PMERR_INV_PATTERN_ATTR	An invalid pattern symbol attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.

PMERR_INV_PATTERN_REF_PT_ATTR	An invalid reftpoint attribute value was specified.
PMERR_INV_PATTERN_SET_ATTR	An invalid pattern set attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_PATTERN_SET_FONT	An attempt was made to use an unsuitable font as a pattern set.
PMERR_INV_PICK_APERTURE_OPTION	An invalid options parameter was specified with GpiSetPickApertureSize.
PMERR_INV_PICK_APERTURE_POSN	An invalid pick aperture position was specified.
PMERR_INV_PICK_APERTURE_SIZE	An invalid size parameter was specified with GpiSetPickApertureSize.
PMERR_INV_PLAY_METAFILE_OPTION	An invalid option parameter was specified with GpiPlayMetaFile.
PMERR_INV_PRIMITIVE_TYPE	An invalid primitive type parameter was specified with GpiSetAttrs or GpiQueryAttrs.
PMERR_INV_PS_SIZE	An invalid size parameter was specified with GpiCreatePS or GpiSetPS.
PMERR_INV_PUTDATA_FORMAT	An invalid format parameter was specified with GpiPutData.
PMERR_INV_QUERY_ELEMENT_NO	An invalid start parameter was specified with DevQueryCaps .
PMERR_INV_RECT	An invalid rectangle parameter was specified.
PMERR_INV_REGION_CONTROL	An invalid control parameter was specified with GpiQueryRegionRects.
PMERR_INV_REGION_MIX_MODE	An invalid mode parameter was specified with GpiCombineRegion.
PMERR_INV_REPLACE_MODE_FUNC	An attempt was made to issue GpiPutData with the editing mode set to SEGEM_REPLACE.
PMERR_INV_RESERVED_FIELD	An invalid reserved field was specified.
PMERR_INV_RESET_OPTIONS	An invalid options parameter was specified with GpiResetPS.
PMERR_INV_RGBCOLOR	An invalid rgb color parameter was specified with GpiQueryNearestColor or GpiQueryColor.

PMERR_INV_SCAN_START	An invalid scanstart parameter was specified with a bit-map function.
PMERR_INV_SEG_ATTR	An invalid attribute parameter was specified with GpiSetSegmentAttrs, GpiQuerySegmentAttrs, GpiSetInitialSegmentAttrs, or GpiQueryInitialSegmentAttrs.
PMERR_INV_SEG_ATTR_VALUE	An invalid attribute value parameter was specified with GpiSetSegmentAttrs or GpiSetInitialSegmentAttrs.
PMERR_INV_SEG_NAME	An invalid segment identifier was specified.
PMERR_INV_SEG_OFFSET	An invalid offset parameter was specified with GpiPutData.
PMERR_INV_SEGLEN	An order length exceeds the remaining segment length in the input PIF.
PMERR_INV_SETID	An invalid setid parameter was specified.
PMERR_INV_SHARPNESS_PARM	An invalid sharpness parameter was specified with GpiPolyFilletSharp.
PMERR_INV_STOP_DRAW_VALUE	An invalid value parameter was specified with GpiSetStopDraw.
PMERR_INV_TRANSFORM_TYPE	An invalid options parameter was specified with a transform matrix function.
PMERR_INV_TYPE	Invalid file-type parameter.
PMERR_INV_USAGE_PARM	An invalid options parameter was specified with GpiCreateBitmap.
PMERR_INV_VIEWING_LIMITS	An invalid limits parameter was specified with GpiSetViewingLimits.
PMERR_INV_VIEWLIM	A set viewing limits order has an inconsistent mask and order length in the input PIF.
PMERR_INV_XFORM	A set (default) viewing transform order has an inconsistent mask and order length in the input PIF.
PMERR_INVALID_APPL	Attempted to start an application whose type is not recognized by OS/2.
PMERR_INVALID_ARRAY_COUNT	An array has an invalid count, that is, less than or equal to zero.
PMERR_INVALID_ARRAY_SIZE	A control data type array size is invalid.

PMERR_INVALID_ASCIIIZ	The profile string is not a valid zero-terminated string.
PMERR_INVALID_ATOM	The specified atom does not exist in the atom table.
PMERR_INVALID_ATOM_NAME	An invalid atom name string was passed.
PMERR_INVALID_BUNDLE_TYPE	An invalid bundle type was passed.
PMERR_INVALID_CHARACTER_INDEX	On WinNextChar or WinPrevChar , a character index is invalid, that is, it is less than 1 or is greater than the string length+1.
PMERR_INVALID_CONTROL_DATATYPE	An invalid control data type was specified.
PMERR_INVALID_DATATYPE	An invalid data type was specified.
PMERR_INVALID_DST_CODEPAGE	The destination code page parameter is invalid.
PMERR_INVALID_ERRORINFO_HANDLE	On WinFreeErrorInfo , the ERRINFO is not the handle of an ERRINFO structure, that is, it was not created by WinGetErrorInfo .
PMERR_INVALID_FLAG	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.
PMERR_INVALID_FREE_MESSAGE_ID	An invalid message identifier was specified. The call has completed by assuming the message parameter and reply data types to be ULONG .
PMERR_INVALID_GROUP_HANDLE	An invalid program-group handle was specified.
PMERR_INVALID_HACCEL	An invalid accelerator-table handle was specified.
PMERR_INVALID_HAPP	The application handle passed to WinTerminateApp does not correspond to a valid session.
PMERR_INVALID_HATOMTBL	An invalid atom-table handle was specified.
PMERR_INVALID_HEAP_POINTER	An invalid pointer was found within the heap.
PMERR_INVALID_HEAP_SIZE	Invalid data was found within the heap.
PMERR_INVALID_HEAP_SIZE_PARM	Invalid data was found within the heap.
PMERR_INVALID_HEAP_SIZE_WORD	Invalid data was found within the heap.
PMERR_INVALID_HENUM	An invalid enumeration

	handle was specified.
PMERR_INVALID_HHEAP	An invalid heap handle was specified.
PMERR_INVALID_HMQ	An invalid message-queue handle was specified.
PMERR_INVALID_HPTR	An invalid pointer handle was specified.
PMERR_INVALID_HSTRUCT	An invalid (null) structure handle was specified.
PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_INI_FILE_HANDLE	An invalid initialization-file handle was specified.
PMERR_INVALID_INTEGER	The specified atom is not a valid integer atom.
PMERR_INVALID_INTEGER_ATOM	The specified atom is not a valid integer atom.
PMERR_INVALID_MESSAGE_ID	A message identifier is invalid.
PMERR_INVALID_NUMBER_OF_PARMS	The number of parameters is invalid.
PMERR_INVALID_NUMBER_OF_TYPES	The function call has an invalid number (zero) of types.
PMERR_INVALID_PARAMETER	An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32,768 to +32,767 cannot be converted to a SHORT, and a negative number cannot be converted to a ULONG or USHORT.
PMERR_INVALID_PARAMETER_TYPE	A parameter type is invalid for a bundle mask.
PMERR_INVALID_PARAMETERS	An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32 768 to +32 767 cannot be converted to a SHORT , and a negative number cannot be converted to a ULONG or USHORT .
PMERR_INVALID_PARAMETERS	
PMERR_INVALID_PARM	A parameter to the function contained invalid data.
PMERR_INVALID_PROGRAM_HANDLE	An invalid program handle was specified.
PMERR_INVALID_SESSION_ID	The specified session identifier is invalid. Either zero (for the application's own session) or a valid identifier must be specified.

PMERR_INVALID_SRC_CODEPAGE	The source code page parameter is invalid.
PMERR_INVALID_STRING_PARM	The specified string parameter is invalid.
PMERR_INVALID_SWITCH_HANDLE	An invalid Window List entry handle was specified.
PMERR_INVALID_TARGET_HANDLE	An invalid target program-group handle was specified.
PMERR_INVALID_TITLE	The specified program or group title is too long or contains invalid characters.
PMERR_INVALID_TYPE_FOR_LENGTH	The data type for a control length is invalid.
PMERR_INVALID_TYPE_FOR_MPARAM	The message parameter type for a control MPARAM is invalid, that is, not mparam1, mparam2 or mreply.
PMERR_INVALID_TYPE_FOR_OFFSET	The data type for a control offset is invalid.
PMERR_INVALID_WINDOW	The window specified with a Window List call is not a valid frame window.
PMERR_KERNING_NOT_SUPPORTED	Kerning was requested on GpiCreateLogFont call to a presentation space associated with a device context that does not support kerning.
PMERR_LABEL_NOT_FOUND	The specified element label did not exist.
PMERR_MATRIX_OVERFLOW	An internal overflow error occurred during matrix multiplication. This can occur if coordinates or matrix transformation elements (or both) are invalid or too large.
PMERR_MEMORY_ALLOC	An error occurred during memory management.
PMERR_MEMORY_ALLOCATION_ERR	An error occurred during memory management.
PMERR_MEMORY_DEALLOCATION_ERR	An error occurred during memory management.
PMERR_METAFILE_IN_USE	An attempt has been made to access a metafile that is in use by another thread.
PMERR_METAFILE_INTERNAL_ERROR	An internal inconsistency has been detected during metafile unlock processing.
PMERR_METAFILE_LIMIT_EXCEEDED	The maximum permitted metafile size limit was exceeded during metafile recording.
PMERR_MSG_QUEUE_ALREADY_EXISTS	An attempt to create a message queue for a thread

	failed because a message queue already exists for the calling thread.
PMERR_MSGID_TOO_SMALL	The message identifier specified is too small.
PMERR_NEGATIVE_STRCOND_DIM	A negative array dimension was passed for a data type length.
PMERR_NO_BITMAP_SELECTED	An attempt has been made to operate on a memory device context that has no bit map selected.
PMERR_NO_CURRENT_ELEMENT	An attempt has been made to issue <code>GpiQueryElementType</code> or <code>GpiQueryElement</code> while there is no currently open element.
PMERR_NO_CURRENT_SEG	An attempt has been made to issue <code>GpiQueryElementType</code> or <code>GpiQueryElement</code> while there is no currently open segment.
PMERR_NO_FILL	No flood fill occurred because either the starting point color was the same as the input color when a boundary fill was requested, or the starting point color was not the same as the input color when a surface fill was requested.
PMERR_NO_METAFILE_RECORD_HANDLE	The metafile record handle was not found during metafile recording, or DevEscape (<code>DEVESC_STARTDOC</code>) was not issued when drawing to a <code>OD_QUEUED</code> device context with a <i>pszDataType</i> field of <code>PM_Q_STD</code> .
PMERR_NO_MSG_QUEUE	
PMERR_NO_PALETTE_SELECTED	An attempt to realize a palette failed because no palette was previously selected into the Presentation Space.
PMERR_NO_SPACE	The limit on the number of Window List entries has been reached with WinAddSwitchEntry .
PMERR_NOT_CREATED_BY_DEVOPENDC	An attempt has been made to destroy a device context using DevCloseDC that was not created using DevOpenDC .
PMERR_NOT_CURRENT_PL_VERSION	An unexpected data format was found in the initialization file.
PMERR_NOT_DRAGGING	A drag operation is not in progress at this time.

PMERR_NOT_IN_A_PM_SESSION	An attempt was made to access function that is only available from PM programs from a non-PM session.
PMERR_NOT_IN_AREA	An attempt was made to end an area using GpiEndArea or during segment drawing while not in an area bracket.
PMERR_NOT_IN_DRAW_MODE	An attempt was made to issue GpiSavePS or GpiRestorePS while the drawing mode was not set to DM_DRAW.
PMERR_NOT_IN_ELEMENT	An attempt was made to end an element using GpiEndElement or during segment drawing while not in an element bracket.
PMERR_NOT_IN_IDX	The application name, key-name or program handle was not found.
PMERR_NOT_IN_IMAGE	An attempt was made to end an image during segment drawing while not in an image bracket.
PMERR_NOT_IN_PATH	An attempt was made to end a path using GpiEndPath or during segment drawing while not in a path bracket.
PMERR_NOT_IN_RETAIN_MODE	An attempt was made to issue a segment editing element function that is invalid when the actual drawing mode is not set to retain .
PMERR_NOT_IN_SEG	An attempt was made to end a segment using GpiCloseSegment while not in a segment bracket.
PMERR_NOT_SELF_DESCRIBING_DTYP	A data type is not self-describing.
PMERR_OK	No errors occurred.
PMERR_OPENING_INI_FILE	Unable to open initialization file (due to lack of disk space for example).
PMERR_ORDER_TOO_BIG	An internal size limit was exceeded while converting orders from short to long format during GpiPutData processing. An order was too long to convert.
PMERR_OWN_SET_ID_REFS	An attempt to unload a font failed because the setid is still being referenced.
PMERR_PALETTE_BUSY	An attempt has been made to reset the owner of a palette when it was busy.
PMERR_PALETTE_SELECTED	Color palette operations

PMERR_PARAMETER_OUT_OF_RANGE	cannot be performed on a presentation space while a palette is selected.
PMERR_PATH_INCOMPLETE	The value of a parameter was not within the defined valid range for that parameter.
PMERR_PATH_LIMIT_EXCEEDED	An attempt was made to open or close a segment either directly or during segment drawing, or to issue GpiAssociate while there is an open path bracket.
PMERR_PATH_UNKNOWN	An internal size limit was exceeded during path or area processing.
PMERR_PEL_IS_CLIPPED	An attempt was made to perform a path function on a path that did not exist.
PMERR_PEL_NOT_AVAILABLE	An attempt was made to query a pel that had been clipped using GpiQueryPel.
PMERR_PRINTER_DD_NOT_DEFINED	An attempt was made to query a pel that did not exist in GpiQueryPel (for example, a memory device context with no selected bit map).
PMERR_PRINTER_QUEUE_NOT_DEFINED	The Presentation Manager device driver has not been defined.
PMERR_PRN_ADDR_IN_USE	The spooler queue for the printer has not been defined.
PMERR_PRN_ADDR_NOT_DEFINED	A printer is already defined on the port.
PMERR_PRN_NAME_NOT_DEFINED	The printer port has not been defined.
PMERR_PROLOG_ERROR	The printer has not been defined.
PMERR_PS_BUSY	A prolog error was detected during drawing. Segment prologs are used internally within retained segments and also appear in metafiles. This error can also arise from an End Prolog order that is outside a prolog.
PMERR_PS_IS_ASSOCIATED	An attempt was made to access the presentation space from more than one thread simultaneously.
PMERR_PS_NOT_ASSOCIATED	An attempt was made to destroy a presentation or associate a presentation space that is still associated with a device context.
PMERR_PS_NOT_ASSOCIATED	An attempt was made to access a presentation space that is not associated with a

PMERR_QUEUE_TOO_LARGE	device context. An attempt to create a message queue has failed because the value specified for the size of the message queue is too large.
PMERR_RASTER_FONT	A request was made for the outline of a bit-map font. Outlines can only be returned for vector font characters.
PMERR_REALIZE_NOT_SUPPORTED	An attempt was made to create a realizable logical color table on a device driver that does not support this function.
PMERR_REGION_IS_CLIP_REGION	An attempt was made to perform a region operation on a region that is selected as a clip region.
PMERR_RESOURCE_DEPLETION	An internal resource depletion error has occurred.
PMERR_RESOURCE_NOT_FOUND	The specified resource identity could not be found.
PMERR_SEG_AND_REFSEG_ARE_SAME	The segid and refsegid specified with GpiSetSegmentPriority were the same.
PMERR_SEG_CALL_STACK_EMPTY	A call stack empty condition was detected when attempting a pop function during GpiPop or segment drawing.
PMERR_SEG_CALL_STACK_FULL	A call stack full condition was detected when attempting to call a segment using GpiCallSegmentMatrix, attempting to preserve an attribute, or during segment drawing.
PMERR_SEG_IS_CURRENT	An attempt was made to issue GpiGetData to a segment that was currently open.
PMERR_SEG_NOT_CHAINED	An attempt was made to issue GpiDrawFrom, GpiCorrelateFrom or GpiQuerySegmentPriority for a segment that was not chained.
PMERR_SEG_NOT_FOUND	The specified segment identifier did not exist.
PMERR_SEG_OVFLOW	The input PIF has more than 1000 called segments. This has overflowed an internal buffer.
PMERR_SEG_STORE_LIMIT_EXCEEDED	The maximum permitted retained segment store size limit was exceeded.

PMERR_SETID_IN_USE	An attempt was made to specify a setid that was already in use as the currently selected character, marker or pattern set.
PMERR_SETID_NOT_FOUND	An attempt was made to delete a setid that did not exist.
PMERR_SMB_OVFLOW	The input PIF has more than 100 symbol sets defined. This has overflowed an internal buffer.
PMERR_SOMDD_IS_ACTIVE	The DSOM daemon is already active.
PMERR_SOMDD_NOT_STARTED	The DSOM daemon failed to start.
PMERR_SOURCE_SAME_AS_TARGET	The direct manipulation source and target process are the same.
PMERR_SPL_CANNOT_OPEN_FILE	Unable to open the file.
PMERR_SPL_DD_NOT_FOUND	The Presentation Manager device driver definition could not be found.
PMERR_SPL_DEVICE_ALREADY_EXISTS	The device already exists.
PMERR_SPL_DEVICE_LIMIT_REACHED	The limit on the number of devices has been reached.
PMERR_SPL_DEVICE_NOT_INSTALLED	The device has not been installed.
PMERR_SPL_DRIVER_ERROR	No Presentation Manager device driver supplied or found.
PMERR_SPL_DRIVER_NOT_INSTALLED	The Presentation Manager device driver has not been installed.
PMERR_SPL_FILE_NOT_FOUND	Unable to find the file.
PMERR_SPL_HARD_NETWORK_ERROR	Hard network error.
PMERR_SPL_INI_FILE_ERROR	Error accessing the initialization file.
PMERR_SPL_INV_DATATYPE	The spool file data type is invalid.
PMERR_SPL_INV_DRIVER_DATATYPE	The data type is invalid for the Presentation Manager device driver.
PMERR_SPL_INV_FORMS_CODE	The forms code for the job is invalid.
PMERR_SPL_INV_HSPL	The spooler handle is invalid.
PMERR_SPL_INV_JOB_ID	The job id is invalid.
PMERR_SPL_INV_LENGTH_OR_COUNT	The length or count is invalid.
PMERR_SPL_INV_PRIORITY	The priority for the job is invalid.

PMERR_SPL_INV_PROCESSOR_DATTYPE	The data type is invalid for the spooler queue processor.
PMERR_SPL_INV_QUEUE_NAME	The spooler queue name is invalid.
PMERR_SPL_INV_TOKEN	The token is invalid.
PMERR_SPL_JOB_NOT_PRINTING	The print job is not printing.
PMERR_SPL_JOB_PRINTING	The print job is already printing.
PMERR_SPL_MANY_QUEUES_ASSOC	More than one queue has been associated with the printer.
PMERR_SPL_NO_CURRENT_FORMS_CODE	There is no current forms code defined to the Presentation Manager device driver.
PMERR_SPL_NO_DATA	No data supplied or found.
PMERR_SPL_NO_DEFAULT_QUEUE	There is no default spooler queue for the printer.
PMERR_SPL_NO_DISK_SPACE	There is not enough free disk space.
PMERR_SPL_NO_FREE_JOB_ID	There is no free job id available.
PMERR_SPL_NO_MEMORY	There is not enough free memory.
PMERR_SPL_NO_QUEUES_ASSOCIATED	A queue has not been associated with the printer.
PMERR_SPL_NO_SUCH_LOG_ADDRESS	The logical address does not exist (that is, it is not defined in the initialization file).
PMERR_SPL_NOT_AUTHORISED	Not authorized to perform the operation.
PMERR_SPL_PRINT_ABORT	The job has already been aborted.
PMERR_SPL_PRINTER_NOT_FOUND	The printer definition could not be found.
PMERR_SPL_PROCESSOR_ERROR	No spooler queue processor supplied or found.
PMERR_SPL_PROCESSOR_NOT_INST	The spooler queue processor has not been installed.
PMERR_SPL_QUEUE_ALREADY_EXISTS	The spooler queue already exists.
PMERR_SPL_QUEUE_ERROR	No spooler queue supplied or found.
PMERR_SPL_QUEUE_NOT_EMPTY	The spooler queue contains print jobs.
PMERR_SPL_QUEUE_NOT_FOUND	The spooler queue definition could not be found.
PMERR_SPL_SPOOLER_NOT_INSTALLED	The spooler is not installed.
PMERR_SPL_STATUS_STRING_TRUNC	The print job status string has

PMERR_SPL_TEMP_NETWORK_ERROR	been truncated. Temporary network error.
PMERR_SPL_TOO_MANY_OPEN_FILES	Too many open files.
PMERR_SPOOLER_QP_NOT_DEFINED	The spooler queue processor has not been defined.
PMERR_START_POINT_CLIPPED	The starting point specified for flood fill is outside the current clipping path or region.
PMERR_STARTDOC_NOT_ISSUED	A request to write spooled output without first issuing a STARTDOC was attempted.
PMERR_STARTED_IN_BACKGROUND	The application started a new session in the background.
PMERR_STOP_DRAW_OCCURRED	Segment drawing or GpiPlayMetaFile was stopped prematurely in response to a GpiSetStopDraw request.
PMERR_TOO_MANY_METAFILES_IN_USE	The maximum number of metafiles allowed for a given process was exceeded.
PMERR_TRUNCATED_ORDER	An incomplete order was detected during segment processing.
PMERR_UNABLE_TO_CLOSE_DEVICE	Unable to close the print device (for example, powered off or offline).
PMERR_UNCHAINED_SEG_ZERO_INV	An attempt was made to open segment with segment identifier zero and the ATTR_CHAINED segment attribute not specified.
PMERR_UNSUPPORTED_ATTR	An unsupported attribute was specified in the attrmask with GpiSetAttrs or GpiQueryAttrs.
PMERR_UNSUPPORTED_ATTR_VALUE	An attribute value was specified with GpiSetAttrs that is not supported.
PMERR_WINDOW_LOCK_OVERFLOW	An overflow occurred for the use count of a window.
PMERR_WINDOW_LOCK_UNDERFLOW	An attempt was made to decrement the use count of a window below zero.
PMERR_WINDOW_NOT_LOCKED	The window specified in WinSendMsg was not locked.
PMERR_WPDSEVER_IS_ACTIVE	The Workplace Shell DSOM Server is already active.
PMERR_WPDSEVER_NOT_STARTED	The Workplace Shell DSOM Server could not be started.
WPERR_INVALID_FLAGS	An invalid flag was specified.
WPERR_INVALID_OBJECTID	An invalid object ID was

WPERR_INVALID_TARGET_OBJECT

specified.

An invalid target object was specified.

Atom Tables

Atom tables enable applications to generate unique identifiers and manage strings. This chapter describes how to use atom tables in PM applications.

About Atom Tables

An *atom table* is an operating system mechanism that an application uses to obtain unique, system-wide identifiers to manage strings efficiently. An application places a string, called an *atom name*, into an atom table and receives a 32-bit integer value, called an *atom*, that the application can use to access that string.

System Atom Table

The *system atom table* is available to all applications. When an application places a string in the system atom table, any application that has the atom name can obtain the atom by querying the system atom table.

An application that defines messages, clipboard-data formats, or dynamic data exchange (DDE) data formats that are intended for use among applications must place the names of the messages or formats in the system atom table. It avoids possible conflicts with messages or formats defined by the system or other applications, and makes the atoms for the messages or formats available to other applications. Applications should use names that are not likely to be used by other applications for other purposes.

Some PM functions enable applications to use atoms in parameters that normally take pointers to strings. For example, WinRegisterClass takes a pointer to a string for its *pszClassName* parameter. WinRegisterClass places the class name string in the system atom table. Afterward, an application can query the system atom table to obtain the atom, then use the atom as the *pszClientClass* parameter of WinCreateStdWindow. This process can save space in the data segment of applications that create many windows of the same private class. Every atom table has a unique handle. An application must obtain the handle before performing any atom operations. To obtain the handle of the system atom table, an application uses WinQuerySystemAtomTable. The atom-table handle returned by this call is used for all other atom functions.

Private Atom Tables

An application can use a *private atom table* to efficiently manage a large number of strings that are used only within the application. The strings in a private atom table, and the resulting atoms, are available only to the application that created the table.

An application that must use the same string in a number of data structures can save data-segment space by using a private atom table. Rather than copying the string into each data structure, the application can place the string in the atom table and use the resultant atom in the data structures. In this way, a string that appears only once in the data segment still can be used many times in the application.

Applications also can use private atom tables to save time when searching for a particular string. To perform a search, an application must place the search string in the atom table only once, then compare the resultant atom with the atoms in the relevant data structures. This usually is faster than doing string comparisons.

Every atom table has a unique handle. An application must obtain the handle before performing any atom operations. To create a private atom table and obtain its handle, an application must use WinCreateAtomTable. The atom-table handle returned by this call must be used for all other atom functions.

An application that no longer needs its private atom table should call `WinDestroyAtomTable` to destroy the table and free the memory that the system allocated for the table.

Atom Types

Applications can use two *types* of atoms: string and integer.

String Atoms

Applications pass null-terminated strings to atom tables and receive *string atoms* (32-bit integers) in return. String atoms have the following properties:

- The maximum number of string atoms allowed is 16K. The values of string atoms are from *0xC000* through *0xFFFF*.
- The maximum amount of data that an atom table can store is 60K. This includes the control data that the operating system uses to manage the atom table (32 bytes for the table plus 8 bytes for each string atom).
- The maximum length of an atom name is 255 characters. A zero-length string is not a valid atom name.
- Case is significant when searching for an atom name in an atom table, and the entire string must match. No substring matching is performed.
- A usage count is associated with each atom name. The count is incremented each time the atom name is added to the table and decremented each time the atom name is deleted from the table. This allows different users of the same string atom to avoid destroying each other's atom names. When the usage count for an atom name equals zero, the system removes the atom and atom name from the table.

Integer Atoms

Integer atoms differ from string atoms as follows:

- Integer atoms are values from *0x0001* through *0xBFFF*. The values of integer atoms and string atoms do not overlap, so the two types of atoms can be intermixed.
- The string representation of an integer atom is *dddd*, where *dddd* are decimal digits. Leading zeros are ignored.
- There is no usage count nor storage overhead associated with an integer atom.

The operating system uses integer atoms to detect whether the same window class name is being defined more than once. The system defines the predefined window class names using integer atoms as constants. When an application registers a window class, the system enters the specified class name in the system atom table. The system then compares the resultant atom with the predefined window-class constants and with the atoms representing the application-defined class names registered earlier. To be able to do this comparison, the system must express the preregistered class names as atoms. By defining the class names as integer atoms, the system ensures that the atoms do not conflict with the string atoms it generates for application-defined class names.

Atom Creation and Usage Count

An application creates an atom by calling `WinAddAtom`, passing an atom-table handle and a pointer to a string. The system searches the specified atom table for the string. If the string already resides in the atom table, the system increments the usage count for the string and

returns the corresponding atom to the application. Repeated calls to add the same atom string return the same atom. If the atom string does not exist in the table when WinAddAtom is called, the string is added to the table, its usage count is set to 1, and a new atom is returned.

An application can retrieve the usage count associated with a given atom using WinQueryAtomUsage. By obtaining the usage count, an application can detect whether other applications, or other threads within the application, are using the same atom.

Atom Deletion

An application calls WinDeleteAtom when it no longer needs to use an atom. WinDeleteAtom reduces the usage count of the corresponding atom by 1. When the usage count reaches zero, the system deletes the atom name from the table.

Atom Queries

An application can find out if a particular string is already in an atom table by using WinFindAtom. WinFindAtom searches the atom table for the specified string and, if the string is there, returns the corresponding atom.

There are two functions that an application can use to retrieve a string from an atom table, provided that the application has the atom corresponding to the desired string. The first, WinQueryAtomLength, returns the length of the string corresponding to the atom. This allows the application to create a buffer of the appropriate size for the string. The second, WinQueryAtomName, retrieves the string and copies it to the buffer.

Atom String Formats

The second parameter to WinAddAtom and WinFindAtom, *pszAtomName*, is a pointer to zero-terminated string. An application can specify this pointer in four ways, as shown in the following table:

Format Name	Description
"!",atom	Points to a string in which the atom is passed indirectly, as a value.
#dddddd	Points to an integer atom specified as a decimal string.
ulong: FFFF(low word)	Passes an atom directly. The atom is in the low word of the <i>pszAtomName</i> parameter. The operating system uses this format to add predefined window classes to the system atom table.
string atom name	The pointer is to a string atom name. Applications typically use this format to add an atom string to an atom table and receive an atom in return.

The *"!",atom* and *ulong: FFFF(low word)* formats are useful when incrementing the usage count of an existing atom for which the original atom string is not known. For example, the system clipboard manager uses the *ulong: FFFF(low word)* format to increment the usage count of each clipboard-format atom when that format is placed on the clipboard. By using this format, the atom is not destroyed even if the original user of the atom deletes it, because the usage count still shows that the clipboard is using the atom.

Using Atom Tables

This section explains how to create unique window-message atoms, dynamic data exchange (DDE) formats and a clipboard format.

Creating Unique Window-Message Atoms

You must create atoms for your application-defined window messages if other applications are likely to recognize those messages. For example, your application might communicate with another application by using an agreed-upon message that is not defined by the system. Both applications must use the same string identifier for the shared message type—for example, `OUR_LINK_MESSAGE`. Each time the applications run, they add this string to the system atom table and receive an atom in return. Both applications register the same string in the system atom table, so they both receive the same atom. Then, this atom can be used to identify the message without conflicting with other system-wide message identifiers. A consequence of using atoms to identify a window message is that the message cannot be decoded as a C-language case statement, as usually done, because the value of the atom cannot be known until run time. Instead, you must add a default case that checks the value of the message against the value of the atoms you have registered. The following sample code fragment shows how to add an application-defined message string to the system atom table, then use the resultant atom to broadcast and receive the message:

```
#define IDM_BROADCAST 25

HATOMTBL hatomtblSystem;          /* System atom table handle    */
ATOM atomLinkMessage;             /* Atom message                */

/* Message text */
UCHAR szLinkMessage[] = "OUR_LINK_MESSAGE";

HRESULT EXPENTRY ClientWndProc(HWND hwnd,ULONG msg,
    MPARAM mp1,MPARAM mp2)
{
    /* At create time obtain atom for text message */
    switch (msg)
    {
        case WM_CREATE:
            hatomtblSystem = WinQuerySystemAtomTable();
            atomLinkMessage = WinAddAtom(hatomtblSystem, szLinkMessage);
            return FALSE;

        /* Broadcast text message */
        case WM_COMMAND:
            if (SHORT1FROMMP(mp1) == IDM_BROADCAST)
            {
                WinBroadcastMsg(HWND_DESKTOP, atomLinkMessage,
                    (MPARAM) NULL, (MPARAM) NULL,
                    BMSG_DESCENDANTS | BMSG_POSTQUEUE);
            }
            return FALSE;

        default:

            /* Check for the atom representing "OUR_LINK_MESSAGE" */
            if (msg == atomLinkMessage) return DoOurMessage(...);
            break;
    }

    /* Execute default window procedure */
    return WinDefWindowProc(hwnd,msg,mp1,mp2);
}
```

Creating DDE Formats and a Unique Clipboard Format

Applications that define their own clipboard or DDE formats must register those formats in the system atom table to avoid conflicting with the

predefined formats and any formats used by other applications. The following sample code fragment shows how to register a custom format:

```
#define MAX_BUF_SIZE 128

HAB hab; /* Anchor block handle */
HATOMTBL hatomtblSystem; /* System atom table handle */
ATOM atomFormatID; /* Atom message */
PSZ pszSrc, pszDest; /* String pointers */
BOOL fSuccess;
CHAR szClipString[MAX_BUF_SIZE];
APIRET rc;

/*****
/* Get the handle of the system atom table,
/* then add the format name to the table.
*****/

/* System atom table handle */
hatomtblSystem = WinQuerySystemAtomTable();
/* Register format string */
atomFormatID = WinAddAtom(hatomtblSystem, "SuperCAD_FORMAT")

/*****
/* Obtain data and write data to buffer (szClipString).
*****/

/* Open the clipboard */
if (WinOpenClipbrd(hab))
{
/* Allocate a shared memory object for the text data */
if (!(rc = DosAllocSharedMem(
    (PVOID)&pszDest, /* Pointer to shared memory */
    (PSZ) NULL, /* object */
    (ULONG)strlen( /* Use unnamed shared memory */
        szClipString) + 1, /* Amount of memory */
    PAG_WRITE | /* Allow write access */
    PAG_COMMIT | /* Commit the shared memory */
    OBJ_GIVEABLE))) /* Make pointer giveable */
{
/* Setup the source pointer to point to text */
pszSrc = szClipString;

/* Copy the string to the allocated memory */
while (*pszDest++ = *pszSrc++);

/* Clear old data from the clipboard */
WinEmptyClipbrd(hab);

/* Pass the pointer to the clipboard in custom format.
/* Notice that the pointer must be a ULONG value.
fSuccess = WinSetClipbrdData(hab, /* Anchor block handle */
    (ULONG) pszDest, /* Pointer to text data */
    atomFormatID, /* Custom format ID (atom) */
    CFI_POINTER); /* Passing a pointer */

/* Close the clipboard */
WinCloseClipbrd(hab);
}
}
```

Button Controls

A *button* is a type of control window used to initiate an operation or to set the attributes of an operation. This chapter describes how to create and use buttons in PM applications.

About Button Controls

A button control can appear alone or with a group of other buttons. When buttons are grouped, the user can move from button to button within the group by pressing the Arrow keys. The user also can move among groups by pressing the Tab key.

A user can select a button by clicking it with the mouse, pressing the spacebar when the button has the keyboard focus, or sending a `BM_CLICK` message. In most cases, a button changes its appearance when selected.

A button control is always owned by another window, usually a dialog window or an application's client window. A button control posts `WM_COMMAND` messages or sends `WM_CONTROL` notification messages to its owner when a user selects the button. For further information on messages generated, see [Button Notification Messages](#). The owner window receives messages from a button control and can send messages to the button to alter its position, appearance, and enabled/disabled state.

To use a button control in a dialog window, an application specifies the control in a dialog template in the application's resource-definition file. The application processes button messages in the dialog-window procedure.

An application creates a button control in a client window by calling `WinCreateWindow`, specifying a window class of `WC_BUTTON`, and identifying the client window as the owner of the button control.

Button Types

There are four main *types* of buttons: push buttons, radio buttons, check boxes, and three-state check boxes. A button's type determines how the button looks and behaves.

A radio button, check box, or three-state check box *controls* an operation; a push button *initiates* an operation. For example, a user might set printing options (such as paper size, print quality, and printer type) in a print-command dialog window containing an array of radio buttons and check boxes. After setting the options, the user would select a push button to tell an application that printing should begin (or be canceled). Then, the application would query the state of each check box and radio button to determine the printing parameters.

Push Buttons

A *push button* is a rectangular window typically used to enable the user to start or stop an operation. When selected, a push button control posts a `WM_COMMAND` message to its owner window.

The push button can contain a text string, an icon or mini-icon, or a combination of text and images.

Radio Buttons

A *radio button* is a window with text displayed to the right of a small circular indicator. Each time the user selects a radio button, that button's state toggles between *selected* and *unselected*. This state remains until the next time the user selects the button. An application typically uses radio buttons in groups.

Within a group, usually one button is selected by default, and the user can move the selection to another button by using the cursor keys; however, only one button can be selected at a time. Radio buttons are appropriate if an *exclusive* choice is required from a fixed list of related options. For example, applications often use radio buttons to allow the user to select the screen foreground and background colors. A radio-button control sends `WM_CONTROL` messages to its owner window.

Check Boxes

Check boxes are similar to radio buttons, except that they can offer *multiple-choice* selection as well as individual choice. Check boxes offer the user a fixed list of choices, with the option of selecting more than one, or even all.

Check boxes also toggle application features *on* or *off*. For example, a word processing application might use a check box to let the user turn word wrapping on or off. A check-box control sends WM_CONTROL messages to its owner window.

Three-State Check Boxes

Three-state check boxes are similar to check boxes, except that they can be displayed in *halftone* as well as selected and unselected. An application might use the halftone state to indicate that, currently, the checkbox is not selectable. A three-state check-box control sends WM_CONTROL messages and posts WM_COMMAND messages to its owner window.

Application-defined Buttons

In addition to using the four predefined button-control types, an application can create button controls that appear as defined by the owner window. When they must be drawn or highlighted, these button controls send WM_CONTROL messages with BN_PAINT as the notification code to their owner windows.

Button Styles

The following table describes the button styles an application can use when creating button controls:

Style	Description.
BS_3STATE	Creates a three-state check box (see also BS_CHECKBOX). When the user selects the check box, it sends a WM_CONTROL message to the owner window. The owner should set the check box to the appropriate state: selected, unselected, or halftone.
BS_AUTO3STATE	Creates an auto-three-state check box (see also BS_CHECKBOX). When the user selects the check box, the system automatically sets the check box to the appropriate state: selected, unselected, or halftone.
BS_AUTOCHECKBOX	Creates an auto-check box (see also BS_CHECKBOX). The system automatically toggles the check box between the selected and unselected states each time the user selects the box.
BS_AUTORADIOBUTTON	Creates an auto-radio button (see also BS_RADIOBUTTON). When the user selects an auto-radio button, the system

automatically selects the button and removes the selection from the other auto-radio buttons in the group.

BS_AUTOSIZE	Creates a button that is sized automatically to ensure that the contents fit. Note: The <i>cx</i> or <i>cy</i> parameter of <i>WinCreateWindow</i> must be specified as -1 to implement the autosize feature.
BS_BITMAP	Creates a push button containing a bit map instead of text. This style can only be implemented with BS_PUSHBUTTON.
BS_CHECKBOX	Creates a check box-a small square that has text displayed to its right. When the user selects a check box, the check box sends a WM_CONTROL message to the owner window. The owner window should toggle the check box between selected and unselected states.
BS_DEFAULT	Creates a push button that has a heavy black border. The user can select this push button by pressing the spacebar. This style is useful for letting the user quickly select the most likely set of options in a dialog window. This style is valid only in combination with the BS_PUSHBUTTON style or the PUSHBUTTON statement in a resource-definition file.
BS_HELP	Creates a push button that posts a WM_HELP message (instead of a WM_COMMAND message) to its owner window when the user selects the button. This style is valid only in combination with the BS_PUSHBUTTON style or the PUSHBUTTON statement in a resource-definition file.
BS_ICON	Creates a push button containing an icon instead of text.
BS_MINIICON	Creates a push button containing a mini-icon instead of text.
BS_NOBORDER	Creates a push button that has no border. This style is valid only in combination with the BS_PUSHBUTTON style or the PUSHBUTTON statement in a resource-definition file.
BS_NOCURSORSELECT	Creates an auto-radio button that will not be selected automatically when the user moves the cursor to the button using the cursor-movement keys. This style is valid only in combination with the BS_AUTORADIOBUTTON style or the AUTORADIOBUTTON statement in a resource-definition file.
BS_NOPOINTERFOCUS	Creates a radio button or check box that does not receive the keyboard focus when the user selects it. This style is valid in combination with the BS_AUTORADIOBUTTON, BS_RADIOBUTTON, BS_3STATE, BS_AUTO3STATE, BS_AUTOCHECKBOX, and BS_CHECKBOX styles, or the AUTORADIOBUTTON, RADIOBUTTON, AUTOCHECKBOX, or CHECKBOX statements in a resource-definition file.
BS_NOTEBOOKBUTTON	Creates a notebook button, which is identical to a pushbutton except that when it is created as a child of a notebook page it becomes a button in the common button area of the notebook page. If the button is not in a notebook page it will be indistinguishable from a

	pushbutton.
BS_PUSHBUTTON	Creates a push button-a round-cornered rectangle with text displayed inside it. When selected, the push button posts a WM_COMMAND message to its owner window.
BS_RADIOBUTTON	Creates a radio button-a small circle that has text displayed to its right. Radio buttons usually are used in groups of related, but exclusive, choices. When the user selects a radio button, the button sends a WM_CONTROL message to its owner window. The user should select the button and remove the selection from the other radio buttons in the group.
BS_SYSCOMMAND	Creates a button that posts a WM_SYSCOMMAND message (instead of a WM_COMMAND message) to the owner window when the user selects the button. This style is valid only in combination with the BS_PUSHBUTTON style or the PUSHBUTTON statement in a resource-definition file.
BS_TEXT	Creates a push button containing both text and icons/mini-icons.
BS_USERBUTTON	Creates a user-defined button that sends a WM_CONTROL message to the owner window when the button needs to be drawn, highlighted, or disabled. A user-defined button also posts WM_COMMAND messages to the owner window when the user selects the button.

Default Button Behavior

Following are the messages processed by the predefined button-control window class (WC_BUTTON). Each message is described in terms of how a button control responds to that message.

Message	Description
BM_AUTOSIZE	Causes the buttons in a new-style notebook to automatically size to fit their contents.
BM_CLICK	Sends a WM_BUTTON1DOWN and WM_BUTTON1UP message to itself to simulate a user button selection.
BM_QUERYCHECK	Returns the checked state of the button.
BM_QUERYCHECKINDEX	Returns the 0-based index to the selected button in a group. Returns -1 if no button in the group is selected or if the button receiving the message is not a radio button or an auto-radio button.
BM_QUERYHILITE	Returns the highlighted state of the button.
BM_SETCHECK	Sets the checked state of the button and returns the previous checked state.

BM_SETDEFAULT	Sets the default button state and redraws the button.
BM_SETHILITE	Sets the highlighted state of the button and returns the previous highlighted state.
WM_BUTTON1DBLCLK	Highlights the button and sends a BN_DBLCLICKED notification code when the button-up message arrives.
WM_BUTTON1DOWN	Sets the button window so it can capture mouse input.
WM_BUTTON1UP	If the button window is set to capture mouse input, and if the mouse pointer is inside the button window when the mouse button is released, this message releases the mouse and sends a notification message to the owner window. If the button is a push button, the push button control posts a WM_COMMAND message; otherwise, the button control sends a WM_CONTROL message with the BN_CLICKED notification code.
WM_CHAR	Sets the button window so it can capture mouse input when the spacebar is pressed; releases the mouse when the spacebar is released. Passes other key messages to the default window procedure.
WM_CREATE	Validates the requested button style and sets the window text.
WM_DESTROY	Frees the memory containing the window's text.
WM_ENABLE	Sent when an application changes the enabled state of a window.
WM_MATCHMNEMONIC	Returns TRUE if <i>mp1</i> matches a mnemonic in the control window's text.
WM_MOUSEMOVE	Sets the default mouse pointer. If the button has the mouse captured, the button's highlighted state changes as the mouse pointer moves in and out of the button boundary.
WM_PAINT	Draws the button according to its style and current state.
WM_QUERYDLGCODE	Returns the DLGC_BUTTON code combined with other DLGC_ codes that designate the button's type.
WM_QUERYWINDOWPARAMS	Returns the requested window parameters.
WM_SETFOCUS	Creates a cursor if the button-control window is receiving the focus. Destroys the cursor if the button-control window is losing the focus.
WM_SETWINDOWPARAMS	Sets the requested window parameters and redraws the button, including the cursor, if the button-control window has the focus.

Button Notification Messages

A button, regardless of its style or type, posts a message to its owner when selected by the user. The message posted by push buttons is ordinarily WM_COMMAND. However, for buttons created with the BS_PUSHBUTTON or BS_USERBUTTON style, the message posted can be changed to WM_HELP or WM_SYSCOMMAND by additionally specifying either the BS_HELP or BS_SYSCOMMAND styles, respectively, when creating the button. A button control that has a style other than BS_PUSHBUTTON or BS_USERBUTTON sends WM_CONTROL messages to its owner when the user selects it.

When the user selects a push button using the mouse pointer, the system automatically highlights the button. The button's window procedure tracks the movement of the pointer until the user releases the button. If the user moves the pointer so that it is outside the button boundary, the system turns off the highlight. The push button control does not post a WM_COMMAND message until the user releases the pointer button, and then, only if the button is released inside the push button boundary. When the owner window receives a WM_COMMAND message from a push button, the low word of the first parameter in the message contains the identifier of the button as specified either in the dialog template or in the WinCreateWindow function when the button was created.

An application should avoid duplicating identifiers for menu items and button controls, because both the items and the controls post identifiers to owner windows as WM_COMMAND messages. However, the application can determine whether a WM_COMMAND message came from a menu or a push button control by looking for the value CMDSRC_MENU or CMDSRC_PUSHBUTTON in the low word of the message's second parameter.

When the user selects any button other than a push button, that button sends a WM_CONTROL message. The application can examine SHORT1FROMMP(*mp1*) in the WM_CONTROL message to find the button identifier, and can examine SHORT2FROMMP(*mp1*) to determine the notification code for the control message. The notification code can be one of the following:

Code	Description
BN_CLICKED	The user selected the button.
BN_DBLCLKED	The user double-clicked the button.
BN_PAINT	A user-defined button needs to be drawn. Buttons with the BS_USERBUTTON style send this notification code to instruct the owner window to draw the button control. The second message parameter of the WM_CONTROL message contains a pointer to a USERBUTTON structure that contains the information necessary for drawing the button.

When the user selects a check box or radio button, the button control sends the WM_CONTROL message with the BN_CLICKED notification code to the owner window. In response, the owner window should set the display state of the button by sending the appropriate message back to the button.

An application need not respond to WM_CONTROL messages sent by an auto-check box or an auto-radio button; the system automatically sets the states of these buttons.

Button States

An application can query and set the highlighted and checked states of its buttons by sending messages to them. An application can obtain the handle of a button by calling WinWindowFromID, using the parent window handle and the identifier of the button. In the case of a dialog window, the parent window would be the dialog window, and the identifier would be the button identifier from the dialog template.

Button-control text is stored as window text. An application can set and retrieve this text by using the WinSetWindowText and WinQueryWindowText functions. To set the size, position, and visibility of a button control, an application uses the standard window functions.

Custom Buttons

An application can customize the appearance of a button by using the BS_USERBUTTON style in combination with other button styles. The owner window receives WM_CONTROL messages for these custom buttons whenever they must be drawn, highlighted, or disabled.

When a button must be drawn, the owner window receives a WM_CONTROL message with the high word of the first parameter equal to BN_PAINT. The second parameter is a pointer to a USERBUTTON structure that contains information the application needs to draw the button.

An application uses the *hwnd* member of the USERBUTTON structure in a call to the WinQueryWindowRect function to find the bounding rectangle for the button. The *hps* member is used as a presentation space for any drawing. The *fsState* member contains flags that tell an application how to draw the button: highlighted, unhighlighted, or disabled. The *fsStateOld* member contains flags that describe the current highlighted, unhighlighted, or disabled state of the button.

Using Button Controls

This section explains how to perform the following tasks:

- Create a dialog template for a button resource
- Create a button for a client window

An application creates a group by setting the WS_GROUP style bit for the first member of the group.

Using Buttons in a Dialog Window

You can define dialog-window buttons as part of a dialog template in a resource-definition file, as shown in the following Resource Compiler source-code fragment.

```
DLGTEMPLATE IDD_BUTTON
BEGIN
    DIALOG  "", 2, 10, 10, 235, 180, WS_VISIBLE, FCF_DLGBORDER
    BEGIN
        AUTORADIOBUTTON "Radio~1", ID_RADIO1, 15, 80, 45, 12, WS_GROUP
        AUTORADIOBUTTON "Radio~2", ID_RADIO2, 15, 60, 45, 12
        AUTORADIOBUTTON "Radio~3", ID_RADIO3, 15, 40, 45, 12
        AUTORADIOBUTTON "Radio~4", ID_RADIO4, 15, 20, 45, 12

        PUSHBUTTON "Button 1", ID_PUSH1, 20, 100, 50, 14, WS_GROUP
        PUSHBUTTON "Button 2", ID_PUSH2, 75, 100, 50, 14, WS_GROUP
        PUSHBUTTON "Button 3", ID_PUSH3, 130, 100, 50, 14, WS_GROUP

        CHECKBOX "Check Box 1", ID_CHECK1, 150, 65, 65, 12, WS_GROUP
        CHECKBOX "no toggle", ID_CHECK2, 150, 40, 58, 12, WS_GROUP
        AUTOCHECKBOX "Check Box 3", ID_CHECK3, 150, 20, 65, 12, WS_GROUP

        DEFPUSHBUTTON "OK", DID_OK, 75, 26, 46, 20, WS_GROUP
    END
END
```

Each button in a dialog window has an identifier (for example, ID_RADIO1) that allows an application to identify the source of the WM_COMMAND and WM_CONTROL messages. An application can use the identifier as the second argument of the WinWindowFromID function to retrieve the button-window handle.

The dialog template also contains the text for each button. For push buttons, this text is displayed in a rectangular box. If the text is too long

to fit in the box, the text is clipped. For radio buttons and check boxes, text is displayed to the right of the button. A user selects the button by clicking either the button or the text itself.

The `WS_GROUP` style identifies the beginning of each new group of buttons. In the preceding example, the four auto-radio buttons are in the same group, and each of the other buttons is in its own group. The auto-radio buttons in the first group can be selected one at a time only. An application must ensure that only one check box in a group is selected at a time. The order in which items can be selected in the group can wrap around from the end of the item list to its beginning.

Notice that the `DEFPUSHBUTTON` style in the preceding example has the identifier `DID_OK`. It is customary to include an **OK** button with this identifier in most dialog windows to provide a uniform user interface. The `DEFPUSHBUTTON` style draws a thick border around a button and allows a user to select the button by pressing the spacebar.

The dialog-window procedure for a dialog window that contains buttons must respond to `WM_COMMAND` and `WM_CONTROL` messages. A common strategy is to use auto-radio buttons and auto-check boxes to let the user set a list of capabilities for a command, and, then, let the user execute the command by choosing an **OK** push button. With this strategy, the dialog-window procedure ignores all `WM_CONTROL` messages that come from auto-radio buttons and auto-check boxes. When the dialog-window procedure receives a `WM_COMMAND` message for the **OK** push button, the procedure should query the auto-radio buttons and auto-check boxes to determine which options have been selected.

Using Buttons in a Client Window

An application can create a button control using an application client window as the owner. The following code fragment shows how an application can use buttons in client windows:

```
#define ID_PBWINDOW 110
HWND hwndButton, hwndClient;

/* Create a button window. */
hwndButton = WinCreateWindow(hwndClient, /* Parent window */
    WC_BUTTON, /* Class window */
    "Test Button", /* Button text */
    WS_VISIBLE | /* Visible style */
    BS_PUSHBUTTON, /* Button style */
    10, 10, /* x, y */
    70, 60, /* cx, cy */
    hwndClient, /* Owner window */
    HWND_TOP, /* Top of z-order */
    ID_PBWINDOW, /* Identifier */
    NULL, /* Control data */
    NULL); /* parameters */
```

Once created in the client window, the button control posts a `WM_COMMAND` message or sends a `WM_CONTROL` message to the client-window procedure. This window procedure should examine the message identifier to determine which button posted or sent the message.

An application that has client-window buttons can move and size the buttons when the client window receives a `WM_SIZE` message. An application can move and size a window by using the `WinSetWindowPos` function. An application can obtain a window handle for a button control by calling the `WinWindowFromID` function, specifying the handle of the parent window and the window identifier for each button.

Creating Buttons with Icons and Icon/Text Combinations

The following styles generate buttons containing images or icons:

- `BS_ICON`
- `BS_MINIICON`
- `BS_BITMAP`

The image or icon is activated by specifying the image ID in the button text string. For example, to load an icon (#define ICON_ID 300) and display it with the button, the button text string is set to "#300".

Where text is to be combined with an image, BS_TEXT is selected. To display an icon (#define ICON_ID 300) with the words "My button", the button text string is set to "#300\tMy button". Notice that "\t" is used to separate text from the image ID.

The following code example creates a customized button with text.

```
// presparm.c -- demonstrates presentation parameters
//              creates a button as a child window
//              and sets its text color

#define INCL_WIN
#define INCL_GPI
#include <os2.h>
#include <string.h>
#include "presparm.h"
#include "migrate.h"

int main ( int argc, char *argv[]);

// Internal function prototypes

MRESULT EXPENTRY MyWindowProc( HWND hwnd, MSGID msg
                               , MPARAM mp1, MPARAM mp2 );
int main ( int argc, char *argv[]);

// global variables

HAB hab; // Anchor block handle

int main ( int argc, char *argv[])
{
    HMQ hmq; // Message queue handle
    HWND hwndFrame; // Frame window handle
    HWND hwndClient; // Client window handle
    QMSG qmsg; // Message from message queue
    ULONG flCreate; // Window creation control flags
    hab = WinInitialize( 0 );
    hmq = WinCreateMsgQueue( hab, 0 );

    WinRegisterClass( hab, "presparm", MyWindowProc, 0L, 0 );

    flCreate = FCF_SYSMENU | FCF_SIZEBORDER | FCF_TITLEBAR |
               FCF_MINMAX | FCF_SHELLPOSITION | FCF_TASKLIST;

    hwndFrame = WinCreateStdWindow( HWND_DESKTOP, WS_VISIBLE, &flCreate,
                                    "presparm", "", 0L, 0, ID_WINDOW, &hwndClient );

    while( WinGetMsg( hab, &qmsg, 0, 0, 0 ) )
        WinDispatchMsg( hab, &qmsg );

    WinDestroyWindow( hwndFrame );

    WinDestroyMsgQueue( hmq );
    WinTerminate( hab );
    return 0;
}

//
MRESULT EXPENTRY MyWindowProc( HWND hwnd, MSGID msg
                               , MPARAM mp1, MPARAM mp2 )
{
    HPS hps; // PS handle
    BTNCDATA btn;

    typedef struct _FOREGOLORPARAM
    {
        ULONG id;
        ULONG cb;

        ULONG ulColor;
    } FORECOLORPARAM;

    typedef struct _FONTPARAM
    {
```

```

        ULONG    id;
        ULONG    cb;
        CHAR     szFontNameSize[20];
    } FONTPARAM;

    struct _PRES                                     // pres. params
    {
        ULONG    cb;                                // length
        FORECOLORPARAM fcparam;                     // foreground color
        FONTPARAM fntparam;                         // font name & size
    } pres;
    static HWND    hwndButton;                      // button window handle
    static POINTL  pt;                              // window size

    switch( msg )
    {
        case WM_CLOSE:
            WinPostMsg( hwnd, WM_QUIT, 0L, 0L );
            return ( (MRESULT) 0 );

        case WM_CREATE:

            // set the foreground color to CLR_RED in
            // the button's presentation parameters
            pres.fcparam.id = PP_FOREGROUND_COLOR_INDEX;
            pres.fcparam.cb = sizeof ( pres.fcparam.ulColor );

            pres.fcparam.ulColor = CLR_RED;

            // set the font used by the button to 12 point Courier
            pres.fntparam.id = PP_FONTNAME_SIZE;
            pres.fntparam.cb = 20;
            strcpy ( pres.fntparam.szFontNameSize, "24.Helv" );

            pres.cb = sizeof ( pres.fcparam ) + sizeof ( pres.fntparam )
            hwndButton = WinCreateWindow ( hwnd          // parent
                , WC_BUTTON                          // class
                , "#300\tNumber One"                  // window text
                , BS_PUSHBUTTON |
                  BS_ICON | BS_TEXT                   // style
                , 100, 100                            // x, y
                , 400, 400                            // cx, cy
                , hwnd                                // owner
                , HWND_TOP                            // sibling
                , 255                                  // ID
                , NULL                                 // ctrl data
                , &pres );                            // pres. params
                                                    // pmasert

            ( hwndButton, hab );
            return (MRESULT)FALSE;

        case WM_SIZE:
            pt.x = (LONG) SHORT1FROMMP ( mp2 );
            pt.y = (LONG) SHORT2FROMMP ( mp2 );
            WinSetWindowPos ( hwndButton, HWND_TOP
                , (SHORT)pt.x / 3
                , (SHORT)pt.y / 2
                , (SHORT)pt.x / 2
                , (SHORT)pt.y / 3
                , SWP_SIZE | SWP_MOVE | SWP_SHOW );

            return (MRESULT)0;

        case WM_PAINT:
            hps = WinBeginPaint ( hwnd , 0 , NULL );
            GpiErase ( hps );
            WinEndPaint ( hps );
            return ( (MRESULT) 0 );

        default:
            return ( WinDefWindowProc( hwnd, msg, mp1, mp2 ) );
    }
    return ( WinDefWindowProc( hwnd, msg, mp1, mp2 ) );

```

Clipboards

The *clipboard* is a small amount of system-managed random-access memory (RAM) used for *user-driven* data exchange. This is in contrast with dynamic data exchange (DDE), which is application driven. While the clipboard stores only *pointers* or *handles* to data, its associated set of functions can be used in applications to move and exchange data. This chapter describes how to use the clipboard in PM applications.

About the Clipboard

The clipboard enables the user to move data in a single application or exchange data between applications. Typically, a user selects data in the application using the mouse or keyboard, and then initiates a cut, copy, or paste operation on that selection.

Descriptions of these operations are in the following table:

Operation	Description
<code>Cut</code>	Deletes the selected data from the application and copies it to the clipboard. Any previous contents of the clipboard are destroyed.
<code>Copy</code>	Copies the selected data to the clipboard. The selection remains unchanged. Previous contents of the clipboard are destroyed.
<code>Paste</code>	Deletes the selected data from the application and replaces it with the contents of the clipboard. The contents of the clipboard are not changed.

An application should not perform any clipboard operations unless the user initiates them explicitly. Other OS/2 features, such as pipes, queues, shared memory, and especially DDE should be used when data exchange is needed without user involvement. For example, an application that continuously passes remotely collected data to a data-analysis application must not use the clipboard. Such an application, instead, should use the other interprocess data-communication capabilities of the operating system.

The data on the clipboard is maintained in memory only. Clipboard data is lost when the computer is turned off.

Shared Memory and the Clipboard

An application must store, in shared memory, text data that is destined for the clipboard. To do so, the application calls the `DosAllocSharedMem` function with the `OBJ_GIVEABLE` attribute to allocate a shared memory object, and then copies the text data to the object. The application passes the clipboard a pointer, which the clipboard uses to access the shared memory object. Clipboard functions use the `CFI_POINTER` flag to indicate text data stored in a shared memory object.

To pass a bit map or metafile to the clipboard, an application passes the clipboard a bit map or metafile handle. The clipboard functions make the bit map or metafile *shareable*. The `CFI_HANDLE` flag is used in clipboard functions to indicate bit map or metafile data.

After closing the clipboard, an application no longer can access the data it passed to the clipboard. Likewise, when an application requests data from the clipboard, it receives a pointer or handle that is good only until the application closes the clipboard. Typically, the application either uses the data immediately before closing the clipboard, or it copies the data to local memory for future use, then closes the clipboard.

Clipboard Operations

An application uses the clipboard when cutting, copying, or pasting data. Typically, an application places data on the clipboard for cut and copy operations and removes data from the clipboard for paste operations. The following paragraphs describe all these clipboard operations.

Cut and Copy Operations

To put data on the clipboard, an application first calls the `WinOpenClipbrd` function to verify that other applications are not trying to retrieve or set clipboard data. The `WinOpenClipbrd` function does not return if another thread has the clipboard open; it waits until either the clipboard is free or there is a message in the message queue of the calling thread. In practice, the `WinOpenClipbrd` function waits until the clipboard is available or until the calling application responds to a message. If the clipboard cannot be opened before a message arrives, the application receives the message, and the `WinOpenClipbrd` function continues to try to open the clipboard. The `WinOpenClipbrd` function does not return until the clipboard is open. However, the application continues to receive messages.

Once an application successfully opens the clipboard, it must remove any previously stored data on the clipboard by calling the `WinEmptyClipbrd` function. If the clipboard is not cleared, writing an existing format on the clipboard replaces the old data in that format with the new data. Old data in other formats remains on the clipboard.

After emptying the clipboard, an application should write its data to the clipboard in as many standard formats as possible. For each format, the application passes the data to the clipboard by calling the `WinSetClipbrdData` function, specifying each data format. The clipboard is not cleared when a new format is written to it; all new data formats coexist on the clipboard until it is cleared by the next clipboard user.

If an application passes `NULL` as the *uiData* parameter of the `WinSetClipbrdData` function, applications must render the data on request.

Finally, when an application finishes writing the clipboard data, it must release the clipboard by calling the `WinCloseClipbrd` function so that other applications can use the clipboard.

Paste Operation

To retrieve data from the clipboard, an application first must call the `WinOpenClipbrd` function to verify that no other applications are trying to retrieve or set the clipboard data.

Once an application successfully opens the clipboard, it calls the `WinQueryClipbrdData` function, specifying a preferred format. If that format is not available (indicated by a `NULL` return from the `WinQueryClipbrdData` function) the application should continue to call `WinQueryClipbrdData` for other possible formats until it either receives the data or runs out of format choices.

If the clipboard contains one of the requested formats, the `WinQueryClipbrdData` function returns a 32-bit integer, the meaning of which depends on the particular format. For text data, the return value is a pointer to a shareable memory object containing the text. For bit map data, the return value is a bit map handle. For metafile data, the return value is a metafile handle.

It is important that an application use the `WinCloseClipbrd` function to close the clipboard as soon as possible so that other applications can access it.

Standard Clipboard-Data Formats

The clipboard can accept data in three standard formats: text, bit map, and metafile. Applications can either use these formats or create their own private formats.

All PM applications can access the clipboard, so applications can copy to the clipboard the same selection of data in many different formats. For example, a word processor that supports multiple fonts might write the same selection of text to the clipboard in three different formats: straight text, *rich* text, and metafile. Then, another application (pasting from the clipboard) could choose the appropriate format.

Applications can use the following constants to specify the standard clipboard-data formats:

Format	Description
<code>CF_BITMAP</code>	Specifies that the data in the clipboard is a bit map.
<code>CF_DSPBITMAP</code>	Specifies that the data in the clipboard is a bit map representation of a private-data format. The clipboard viewer uses this format to display a private format.
<code>CF_DSPMETAFILE</code>	Specifies that the data in the clipboard is a metafile representation of a private-data format. The clipboard viewer uses this format to display a private format.
<code>CF_DSPTEXT</code>	Specifies that the data in the clipboard is a text representation of a private-data format. The clipboard viewer uses this format to display a private format.
<code>CF_METAFILE</code>	Specifies that the data in the clipboard is a metafile.
<code>CF_TEXT</code>	Specifies that the data in the clipboard is an array of text characters. These characters can include <i>newline</i> characters to indicate line breaks. The NULL character indicates the end of the text data.

Private Clipboard-Data Formats

Applications that use the clipboard to move data within the documents of the application can use private clipboard-data formats when standard formats are insufficient for representing clipboard data. For example, a word processor might have a rich-text format that contains font and style information in addition to the usual text characters. Clearly, if the word processor uses the clipboard to support cut, copy, and paste operations for moving data in its documents, a standard text format will be inadequate.

In such case, the word processor should write at least two formats to the clipboard for each cut or copy operation: a standard text format representing the text of the current selection and a private rich-text format representing the true state of the selection. If the word processor performs a paste operation by using clipboard data, it can use the rich-text format to retain all formatting. If another application requests the same data, it can use the standard-text format if it does not recognize the private format. Also, the word processor should be able to render data in `CF_BITMAP` and `CF_METAFILE` formats for painting and drawing applications.

Format Identification Number

Each private format must have a unique identification number. To obtain an identification number, the application registers the name of the private format in the system atom table. The system assigns a unique identification number for the format name. Other applications having access to the format name can query the system atom table for the format identification number.

An application can interpret its own private formats and request them from the clipboard for cutting and pasting its own data. Other applications that know the private-format identification number also can interpret the formatted data.

Display Formats

The OS/2 operating system provides three standard display formats for applications that use private formats: CF_DSPTEXT, CF_DSPBITMAP, and CF_DSPMETAFILE. These formats correspond to the standard text, bit map, and metafile formats, with the exception that they are intended for use only by the clipboard viewer. An application that uses a private format should write one of the DSP formats that approximates the appearance of the private data so that the clipboard viewer can display the data regardless of the format. For example, a word processor using the rich-text format also would write a CF_DSPBITMAP formatted picture of the selected text that contains all the type fonts and styles.

Notice that you can choose delayed rendering for DSP formats because there might not always be a clipboard viewer active on the screen. With delayed rendering, an application actually does not render the format unless it is requested to do so.

Delayed Rendering

An application can pass NULL as the *uiData* parameter of the WinSetClipbrdData function instead of a pointer or a handle. This indicates that the data is rendered only when another application requests it from the clipboard. This is useful if an application supports several clipboard formats that are time-consuming to render. With delayed rendering, an application can send NULL handles for each clipboard format that it supports and render individual formats only when the format actually is requested from the clipboard. An application can either write data for standard formats or choose delayed rendering for more complex formats.

When an application uses delayed rendering for one or more of its clipboard formats, it must become the clipboard owner. As long as the application is the clipboard owner, it receives a WM_RENDERFMT message whenever a request is received by the clipboard for a format using delayed rendering. When the application receives such a message, it renders the data and passes the pointer or handle to the clipboard by calling the WinSetClipbrdData function. The rules for shared-memory access for rendered data are the same as those for standard clipboard data. This simply is a delayed execution of the operation that occurs if the data does not have delayed rendering.

The clipboard owner, with one or more delayed-rendering formats on the clipboard, receives a WM_RENDERALLFMITS message just before the clipboard-owner application terminates. This ensures that the application renders all of its data before terminating.

Clipboard Viewer

A window can become a clipboard viewer and display the current contents of the clipboard. The clipboard viewer is informed whenever the clipboard contents change. Typically, the clipboard viewer is a window that can draw the standard clipboard formats. The clipboard viewer is a convenience for the user; it does not have any effect on the data-transaction functions of the clipboard.

To create a clipboard viewer, an application calls WinSetClipbrdViewer, specifying the window in which the clipboard data will be displayed. Usually this is the client window of an application. There can be only one clipboard viewer at any time in the system, so setting a clipboard viewer replaces any previous clipboard viewer. The WinQueryClipbrdViewer function receives the handle to the current clipboard viewer so that the application can reset it when finished with the clipboard viewer.

Once a window becomes the clipboard viewer, it receives WM_DRAWCLIPBOARD messages whenever the contents of the clipboard change. The window should respond to these messages by drawing the contents of the clipboard.

The clipboard viewer displays all the standard formats and should process CFI_OWNERDISPLAY items by sending the appropriate message to the clipboard owner.

The clipboard viewer cannot display private-format data. For this reason, an application that writes private-format data to the clipboard also must write the data in one of the three standard-display formats: CF_DSPTEXT, CF_DSPBITMAP, or CF_DSPMETAFILE.

If a standard format is not provided in addition to the private formats, the clipboard owner must draw the clipboard data in the clipboard-viewer window. An application uses the CFI_OWNERDRAW flag to identify clipboard data that the clipboard owner draws. When the clipboard viewer encounters data with the CFI_OWNERDRAW flag set, it sends WM_PAINTCLIPBOARD messages to the clipboard owner whenever the data must be drawn, scrolled, or sized.

The clipboard viewer determines the attributes of a particular clipboard format by calling the WinQueryClipbrdFmtInfo function. The identity of the current owner is found by calling the WinQueryClipbrdOwner function.

Clipboard Owner

The *clipboard owner* is any application window connected to the clipboard data. Following are situations in which an application would call WinSetClipbrdOwner to become the clipboard owner:

- The application calling WinSetClipbrdData passes a NULL pointer or handle to the clipboard, indicating that the application renders the data in a particular format on request. As a result, the system sends rendering requests to the current clipboard owner.
- The application calling WinSetClipbrdData passes data with the CFI_OWNERFREE attribute, indicating that the application frees memory for data when the clipboard is emptied. As a result, the system sends owner-free requests to the current clipboard owner.
- The application calling WinSetClipbrdData passes data with the CFI_OWNERDISPLAY attribute, indicating that the owner application draws the data in the clipboard viewer. As a result, the clipboard viewer sends drawing-related requests to the current clipboard owner.

The window specified in the call to the WinSetClipbrdOwner function responds to the following messages:

Message	Description
WM_RENDERFMT	Sent by the system to the clipboard owner when a particular format with delayed rendering must be rendered. The receiver must render the data in the specified format and pass it to the clipboard by calling the WinSetClipbrdData function.
WM_RENDERALLFMTS	Sent by the system to the clipboard owner just before the owner application terminates. The receiver must render the clipboard data in all formats on the clipboard with delayed rendering. It must pass the data for each format to the clipboard by calling the WinSetClipbrdData function.
WM_DESTROYCLIPBOARD	Sent by the system to the clipboard owner when the clipboard is cleared by another application calling the WinEmptyClipbrd function. The receiver must free the memory occupied by any clipboard formats using the CFI_OWNERFREE attribute.
WM_SIZECLIPBOARD	Sent by the clipboard viewer to the clipboard owner when the clipboard contains the data handle with the CFI_OWNERDISPLAY attribute and when the clipboard-viewer changes size. When the clipboard viewer is being destroyed or reduced to an icon, this message is sent with the coordinates of the opposite corners set to (0,0), which permits the owner to free its display resources.
WM_VSCROLLCLIPBOARD	Sent by the clipboard viewer to the clipboard owner when the clipboard contains data with the CFI_OWNERDISPLAY attribute and when an event occurs in the clipboard-viewer scroll bars. The receiver must respond to this message by scrolling the image, invalidating the appropriate area of the clipboard viewer, and updating the slider position.
WM_HSCROLLCLIPBOARD	Sent by the clipboard viewer to the clipboard owner when the clipboard contains data with the CFI_OWNERDISPLAY attribute and when an event occurs in the scroll bars of the clipboard viewer. The receiver must respond to this message by scrolling the image, invalidating the appropriate area of the

clipboard viewer, and updating the slider position.

WM_PAINTCLIPBOARD

Sent by the clipboard viewer to the clipboard owner when the clipboard contains data with the CFI_OWNERDISPLAY attribute and when the clipboard-viewer client area needs repainting. The receiver must respond to this message by painting the requested format (by calling WinGetPS for the window handle of the clipboard viewer).

An application automatically loses ownership of the clipboard when the clipboard data is cleared by the WinEmptyClipbrd function. Ownership is necessary only when data is present on the clipboard. Typically, an application loses ownership when another application places data on the clipboard.

Using the Clipboard

You can use the clipboard functions to perform the following tasks:

- Put data on the clipboard
- Retrieve data from the clipboard
- View data on the clipboard

Putting Data on the Clipboard

The following code fragment shows how an application places text data on the clipboard, how it opens the clipboard, copies the text to a shared memory object, empties the clipboard, and passes the pointer to the clipboard:

```
#define MAXSTR    1024

PSZ  pszSrc, pszDest;
BOOL fSuccess;
CHAR szClipString[MAXSTR];
HAB  hab;

.
. /* Get character string (szClipString). */
.

if (WinOpenClipbrd(hab)) {

    /* Allocate a shared memory object for the text data. */
    if (!(fSuccess = DosAllocSharedMem(
        (PVOID)&pszDest,          /* Pointer to shared memory object */
        NULL,                     /* Use unnamed shared memory      */
        strlen(szClipString)+1, /* Amount of memory to allocate   */
        PAG_WRITE |               /* Allow write access              */
        PAG_COMMIT |             /* Commit the shared memory        */
        OBJ_GIVEABLE))) {          /* Make pointer giveable          */

        /* Set up the source pointer to point to text. */
        pszSrc = szClipString;

        /* Copy the string to the allocated memory. */
        while (*pszDest++ = *pszSrc++);

        /* Clear old data from the clipboard. */
```

```

WinEmptyClipbrd(hab);

/*
 * Pass the pointer to the clipboard in CF_TEXT format. Notice
 * that the pointer must be a ULONG value.
 */

fSuccess = WinSetClipbrdData(hab, /* Anchor-block handle */
    (ULONG) pszDest, /* Pointer to text data */
    CF_TEXT, /* Data is in text format */
    CFI_POINTER); /* Passing a pointer */

/* Close the clipboard. */
WinCloseClipbrd(hab);
}
}

```

Retrieving Data from the Clipboard

The following code fragment shows how to open the clipboard, retrieve data in the requested format, copy the data to local memory, and close the clipboard:

```

PSZ pszClipText, pszLocalText;

if (WinOpenClipbrd(hab)) {
    if (pszClipText = (PSZ) WinQueryClipbrdData(hab, CF_TEXT)) {

        /* Copy text from the shared memory object to local memory. */
        while (*pszLocalText++ = *pszClipText++);
    }
    WinCloseClipbrd(hab);
}

```

Viewing Data on the Clipboard

The following code fragment shows how a sample clipboard viewer responds to the WM_DRAWCLIPBOARD message, drawing text and bit map data in its window. Notice that the code uses the data retrieved from the clipboard before closing the clipboard. An alternative strategy is to copy the data and then close the clipboard. In any case, the original data from the clipboard cannot be used after the clipboard is closed.

```

PSZ    pszText;
HPS    hps;
RECTL  rcl;
HBITMAP hBitmap;
POINTL  ptlDest;

case WM_DRAWCLIPBOARD:
    if (!WinOpenClipbrd(hab))
        return 0;

    hps = WinGetPS(hwnd); /* Get a presentation space for drawing */
    WinQueryWindowRect(hwnd, &rcl); /* Get dimensions of the window */

```

```

if (pszText = (PSZ)WinQueryClipbrdData(hab, CF_TEXT)) {
    WinDrawText(hps,
        -1, /* Null-terminated string */
        pszText, /* The string */
        &rcl, /* Where to put the string */
        CLR_BLACK, /* Foreground color */
        CLR_WHITE, /* Background color */
        DT_CENTER | DT_VCENTER | DT_ERASERECT);
}
else if (hBitmap = (HBITMAP)WinQueryClipbrdData(hab, CF_BITMAP)) {
    ptlDest.x = ptlDest.y = 0;
    WinFillRect(hps, &rcl, CLR_WHITE);
    WinDrawBitmap(hps,
        hBitmap,
        NULL, /* Draws entire bit map */
        &ptlDest, /* Destination */
        CLR_BLACK, /* Foreground color */
        CLR_WHITE, /* Background color */
        DBM_NORMAL); /* Bit map flags */
}

/* Remove rectangle from the update region */
WinValidateRect(hwnd, &rcl, FALSE);
WinReleasePS(hps); /* Release the presentation space.*/
WinCloseClipbrd(hab); /* Close the clipboard. */
return 0;

```

Responding to WM_DRAWCLIPBOARD Message

Combination Box

A *combination box* is two controls in one: an entry field and a list box. This chapter describes how to use *combination-box controls*, also called *combination boxes* and *prompted entry fields*, to let the user choose and edit items from a list in PM applications.

About Combination-Box Controls

Combination-box controls enable the user to enter data by typing in the entry field or by choosing from a list in the list box. A combination-box control automatically manages the interaction between the entry field and the list box. For example, when the user chooses an item in the list box, the combination-box control displays the text for that item in the entry field. Then, the user can edit the text without affecting the item in the list box. When the user types a letter in the entry field, the combination-box control scrolls the list box contents so that items beginning with that letter become visible.

Combination-Box Styles

The following table shows the combination-box styles:

Style Name	Description
CBS_SIMPLE	Creates a simple combination box that always displays its list box. The user can enter and edit text in the entry field or choose items from the list box.
CBS_DROPDOWN	Creates a drop-down combination box that displays its list box only if the user

clicks the drop-down icon at the right end of the entry field. The combination-box control hides the list box when the user clicks the icon a second time. In a drop-down combination box, the user can enter and edit text in the entry field or choose items from the list box.

CBS_DROPDOWNLIST Creates a drop-down-list combination box that is similar to the drop-down combination box, except that the user can choose items only from the list box. The user cannot enter or edit text in the entry field.

For combination boxes that have the CBS_DROPDOWN or CBS_DROPDOWNLIST styles, an application can display the list by using the CBM_SHOWLIST message.

An application can determine whether the list is already showing by using the CBM_ISLISTSHOWING message.

Applications also can use any of the entry-field (EM_) and list-box (LM_) messages with combination boxes. Entry-field messages affect the entry field; list-box messages affect the list box. For example, an application can use the LM_INSERTITEM message to insert items into the list box.

Combination-Box Notification Codes

A combination-box control sends WM_CONTROL messages containing notification codes to its parent window. These notification codes are similar to those sent by entry-field and list-box controls. A combination-box control sends a notification codes to its owner window.

Using Combination-Box Controls

You can create a combination box by using WinCreateWindow or by specifying a COMBOBOX statement in a dialog-window template in a resource file. When creating a combination box using WinCreateWindow, you must specify the predefined class WC_COMBOBOX. If you do not specify a style, the function uses the default styles WS_GROUP, WS_TABSTOP, and WS_VISIBLE.

Container Controls

A *container* control (WC_CONTAINER window class) is a visual component that holds objects. It provides a powerful and flexible component for easily developing products that conform to the Common User Access (CUA) user interface guidelines. This chapter describes the container control component and how to use it in PM applications.

About Container Controls

A container can display objects in various formats and views. Generally speaking, each view displays different information about each object. If a container's data is too large for the window's *work area*, scrolling mechanisms are enabled. The CUA direct manipulation protocol is fully supported, thereby enabling a user to visually drag an object in a container window and drop it on another object or container window. Containers are an integral component of the CUA user interface.

Container Control Functionality

The container control provides multiple views of a container's contents, such as Icon, Name, Text, Tree, and Details views. The container control lets you change container views quickly and easily, display each view with a different font, or vertically split the Details view into two parts so that a user can widen one part to see more information.

Graphical user interface (GUI) support is part of the container control. GUI support allows:

- Direct manipulation
- Multiple selection types: single, extended, and multiple selections
- Multiple selection techniques: marquee, swipe, and first-letter selection
- Multiple selection mechanisms: mouse button 1, mouse button 2, and keyboard augmentation
- Multiple forms of emphasis: selected-state, unavailable-state, in-use, and target emphasis
- Scrolling when a container's work area is not large enough for all the container items to be visible
- Dynamic scrolling to provide visible feedback to show the movement of the container items relative to the position of the scroll box

The container control supports various data types, such as icons or bit maps for the Icon, Name, Tree, and Details views. In the Details view, this includes the ability to use icons or bit maps in column headings as well as in the columns themselves. The container control also supports text in the following situations:

- For container titles in all views
- Beneath icons or bit maps in the Icon view
- To the right of icons or bit maps in the Name and Tree views
- For any column or column heading in the Details view
- For container items in the Text view
- For container items in the Details view, text in date, time, and number format

The container control provides a variety of options for enhancing the performance of the container:

- Direct editing of container control text
- Blank text fields in all views
- Ownerdraw, which enables an application, rather than the container control, to draw the container items
- Automatic reposition mode which is used in the Icon view. The container control provides an automatic reposition mode that repositions the items as a result of inserting, removing, sorting, filtering items, or changing window or font size.
- Arrange message mode that arranges overlapping icons or bit maps so they no longer overlap
- Data caching to efficiently remove items from and insert items into a container as they scroll in and out of view
- Methods for sharing records among multiple containers
- Memory usage optimization

Container Items

Container items can be anything that your application or a user might store in a container. Examples are executable programs, word processing files, graphic images, and database records.

Container item data is stored in the RECORDCORE and MINIRECORDCORE data structures. Both the application and the container have access to the data stored in these records.

The application must allocate memory for each record by using the CM_ALLOCORECORD message.

The maximum number of records is limited only by the amount of memory in the user's computer. The container control does not limit the number of records that a container can have.

Container Views

When a user opens a container, the contents of that container are displayed in a window. A container window can present various views of its contents, and each view can provide different information about its container items. The following table describes the views the container control provides:

View Type	Contents Displayed
Icon view	Displays either icons or bit maps, with text beneath the icons or bit maps, to represent container items. These are called icon/text or bit-map/text pairs. Each icon/text or bit-map/text pair represents one container item. This is the default view.
Name view	Displays either icons or bit maps, with text to the right of the icons or bit maps, to represent container items. These are called icon/text or bit-map/text pairs. Each icon/text or bit-map/text pair represents one container item.
Text view	Displays a simple text list to represent container items.
Tree view	Displays a hierarchical view of the container items. Three types of Tree views are available: Tree text, Tree icon, and Tree name.
Details view	Displays detailed information about each container item. The same type of data is displayed for each container item, arranged in columns. The data in each column can consist of an icon or bit map, text, numbers, dates, or times.

If a text string is not specified for a view in a place where a text string could be used, a blank space is used as a placeholder. For example, if a text string is not placed beneath an icon in the Icon view, a blank space is inserted just as though the text string were there. If this blank space is not a read-only field, the user can put text into the space by editing it directly.

Icon View

The Icon view (CV_ICON attribute) displays icon/text pairs or bit-map/text pairs to represent container items; this is the default view. CV_ICON is an attribute of the CNRINFO data structure's *flWindowAttr* field.

In the Icon view, icon/text pairs and bit-map/text pairs are icons and bit maps, respectively, with one or more lines of text displayed below each icon or bit map. Each line can contain one or more text characters, which are centered below the icon or bit map. The container control does not limit the number of lines or the number of characters in each line.

Generally, the icon or bit map contains an image that depicts the type of container item that it represents. For example, an icon or bit map that represents a bar chart might contain an image of a bar chart.

Because the container control does not support both icons and bit maps in the same view, an application must specify which are used by setting either the CA_DRAWICON attribute or the CA_DRAWBITMAP attribute in the *flWindowAttr* field of the CNRINFO data structure. The default is the CA_DRAWICON attribute. The size of the icon or bit map can be specified in the *sBitmapOrIcon* field of the CNRINFO data structure.

In the Icon view, container items are positioned according to *x*- and *y*-coordinate positions. These are called *workspace coordinates*. You can supply these coordinates for each container item by using the *ptIcon* field of the RECORDCORE data structure.

If you do not specify *x*- and *y*-coordinate positions, the container control places the icons or bit maps at (0,0). However, your application can arrange the icons or bit maps either by sending the CM_ARRANGE message or by setting the CCS_AUTOPOSITION style bit when creating a container. With both of these methods, the container items are arranged in rows, and any coordinates specified in the *ptIcon* field are ignored. As they are arranged each *ptIcon* is updated with its new location.

The container items fill the topmost row until the width of the work area is reached. The items then wrap to form another row immediately below the filled row. This process is repeated until all the container items are positioned in rows. Default spacing is implemented according to the guidelines for the CUA user interface.

If the CCS_AUTOPOSITION style bit is set and the container is displaying the Icon view, container items are arranged automatically, without the CM_ARRANGE message being sent, when:

- The window size changes
- Container items are inserted, removed, sorted, invalidated, or filtered
- The font or font size changes

In all of these cases, container items are arranged the same as when the CM_ARRANGE message is sent. The CCS_AUTOPOSITION style bit is valid only when it is used with the Icon view.

If the CM_ARRANGE message is issued and the container control is not currently displaying the Icon view, the container items are still arranged logically. Nothing changes in the current view; the arrangement of the container items is not visible until the user switches to the Icon view.

Name View

The Name view (CV_NAME attribute) displays icon/text or bit-map/text pairs to represent container items. CV_NAME is an attribute of the CNRINFO data structure's *fWindowAttr* field.

In the Name view, icon/text pairs and bit-map/text pairs are icons and bit maps, respectively, with one or more lines of text displayed to the right of each icon or bit map. Each line can contain one or more text characters, which are left-justified. The container control does not limit the number of lines or the number of characters in each line.

The container control offers the option of flowing or not flowing the container items in the Name view. To *flow* container items means to dynamically arrange them in columns.

Non-Flowed Name View

If the container items are not flowed, the icon/text or bit-map/text pairs are placed in a single column in the leftmost portion of the work area.

Flowed Name View

If the container items are flowed (CV_NAME | CV_FLOW), the container appears. In this case, the container items fill the leftmost column until the depth of the work area is reached. The items then wrap to form another column immediately to the right of the filled column. This process is repeated until all of the container items are positioned in columns.

The width of each column is determined by the widest text string within the column. The height of the work area is determined by the size of the window.

Text View

The Text view (CV_TEXT attribute) displays one or more lines of text to represent container items. CV_TEXT is an attribute of the CNRINFO data structure's *fWindowAttr* field.

Each line can contain one or more text characters, which are left-justified. The container control does not limit the number of lines or the number of characters in each line.

The container control offers the option of flowing or not flowing the container items in the Text view.

Non-Flowed Text View

If the text strings are not flowed, the text for each container item is placed in a single column in the leftmost portion of the work area.

Flowed Text View

If the text strings are flowed (CV_TEXT | CV_FLOW), the container appears. In this case, the text strings fill the leftmost column until the depth of the work area is reached. The text strings then wrap to form another column immediately to the right of the filled column. This process is repeated until all the text strings are positioned in columns.

The width of each column is determined by the widest text string within the column. The height of the work area is determined by the size of the window.

Tree View

The Tree view (CV_TREE attribute) displays container items arranged hierarchically. CV_TREE is an attribute of the CNRINFO data structure's *fWindowAttr* field.

The leftmost items displayed in the Tree view are at the *root level* and are the same items displayed in all the other container views. Items that contain other items are called *parent items*. The items that a parent item contains are called *child items* and can be displayed only in the Tree view. Child items that contain other items serve a dual role: they are the children of their parent item, but they are parent items as well, with children of their own. For example, a parent item might be a book that contains individual child items for its chapters or a folder that contains several reports. The chapters or reports, in turn, could be parent items that contain their own children, such as the major sections of a chapter or report.

If the child items of a parent item are not displayed, the parent item can be *Expanded* to display them as a new branch in the Tree view. Once a parent item has been expanded, it can be *Collapsed* to remove its child items from the display.

You can use the *cxTreeIndent* and *cxTreeLine* fields of the CNRINFO data structure to specify the number of pels that a new branch is to be indented horizontally, and the width of the lines that are used to connect branches of the tree. These lines are displayed only if the CA_TREELINE attribute is specified in the *fWindowAttr* field.

The Tree view has three different types: Tree icon view, Tree text view, and Tree name view. If CV_TREE is specified, the Tree icon view is the default view. If neither CV_ICON, CV_TEXT, or CV_NAME are specified, CV_ICON is assumed.

Tree Icon View and Tree Text View

The Tree icon and Tree text views are identical in every aspect except their appearance on the screen. Container items in the Tree icon view (CV_TREE | CV_ICON) are displayed as either icon/text pairs or bit-map/text pairs. The items are drawn as icons or bit maps with one or more lines of text displayed to the right of each icon or bit map.

Container items in the Tree text view (CV_TREE | CV_TEXT) are displayed as text strings. In both views, the container control does not limit the number of lines of text or the number of characters in each line.

In the Tree icon and Tree text views, a parent item is expanded by selecting the Collapsed icon/bit map, which is displayed to the left of the parent item.

The Collapsed icon/bit map should contain some visible indication that the item can be expanded. The default Collapsed bit map that is

provided by the container control uses a plus sign (+) to indicate that more items, the children of this parent, can be added to the view.

When the child items of a parent item are displayed, the Collapsed icon/bit map to the left of that parent item changes to an Expanded icon/bit map. Just as the Collapsed icon/bit map provides a visible indication that an item can be expanded, so should the Expanded icon/bit map indicate that an item can be collapsed. The default Expanded bit map provided by the container control uses a minus sign (-) to indicate that the child items of this parent can be subtracted from the view. If any of the child items have children of their own, a Collapsed or Expanded icon/bit map is displayed to their immediate left as well.

To display your own Collapsed and Expanded icons or bit maps, specify their handles by using the *hptrCollapsed* and *hptrExpanded* fields of the CNRINFO data structure for icons, and the *hbmCollapsed* and *hbmExpanded* fields for bit maps. Also, you can use the *slTreeBitmapOrIcon* field to specify the size, in pels, of these Collapsed and Expanded icons and bit maps.

Tree Name View

Container items in the Tree name view (CV_TREE | CV_NAME) are displayed as either icon/text pairs or bit-map/text pairs. Similar to the Tree icon view, the items are drawn as icons or bit maps with one or more lines of text displayed to the right of each icon or bit map. The container control does not limit the number of lines or the number of characters in each line of text.

Unlike the Tree icon view, however, separate Collapsed and Expanded icons/bit maps are not used. Instead, if an item is a parent, the icon or bit map that represents that item contains the same type of visible indication that is placed in a separate icon/bit map in the Tree icon view to show that an item can be collapsed or expanded. In this way, the icon or bit map that represents the parent item can serve a dual purpose and thus preserve space on the screen, an important consideration if the text strings used to describe items become too long.

The container control does not provide default icons or bit maps for the Tree name view. To display your own Collapsed and Expanded icons or bit maps, specify their handles using the *hptrCollapsed* and *hptrExpanded* fields of the TREEITEMDESC data structure for icons, and the *hbmCollapsed* and *hbmExpanded* fields for bit maps. Also, you can use the *slBitmapOrIcon* field of the CNRINFO data structure to specify the size, in pels, of these Collapsed and Expanded icons and bit maps.

Details View

The Details view (CV_DETAIL attribute) of the container control can display the following data types to represent container item: icons or bit maps, text, numbers, dates, and times. CV_DETAIL is an attribute of the CNRINFO data structure's *flWindowAttr* field.

The data is arranged in columns, which can have headings. Each column can contain data that belongs to only one of the valid data types. Column headings can contain text, icons, or bit maps.

The width of each column can be explicitly specified in the *cxWidth* field of the FIELDINFO data structure. If a column width is not specified, it is determined by the widest entry in the column.

Columns can be inserted or removed dynamically. All of the columns in a given row belong to a single container item; selecting the data portion of a row selects the entire row, not just the individual column.

Details view column headings and data can be top- or bottom-justified or vertically centered, as well as left- or right-justified or horizontally centered. In addition, horizontal separator lines can be specified between the column headings and the data; vertical separator lines can be placed between columns.

Determining the Width of a Column in a Details View

There are instances when you might want to determine the width of a column in the Details view. A function has been added to the container control to allow you to determine the width of the data in a column. You can then compute the width of the entire column by adding the width of the data to the left and right margins of the column. To determine the width of a column:

1. Define an attribute with a value of 0x0200 and give it a name such as CMA_DATAWIDTH.
2. Issue the CM_QUERYDETAILFIELDINFO message with the following values:

- a. Provide a pointer to the FIELDINFO data structure in *param1*.
 - b. Specify your attribute (see step 1) in *param2*.
 - c. Request a return value with a type of LONG, not PFIELDINFO, to retrieve the width of the column in the FIELDINFO data structure to which you are pointing. The value returned is the width of the data (text, icon, or bit map) in this column.
3. Use GpiQueryFontMetrics to query the average character width of the font used by the container. This value will be used to calculate the total column width.
4. Multiply 3 by the average character width and add this to the data width returned from step 2 for all columns except the following:
 - The first and last columns in each split window. In these cases, multiply 2.5 by the average character width and add the column data width returned from step 2.
 - The only other special case is where there is only 1 column in either the left or right split windows. In this case, you would multiply 2 by the average character width and add the column data width returned from step 2.
5. The value returned is the total width of the column.

Split Bar Support for the Details View

A split bar enables the application to split the container window vertically between two column boundaries. This function is available only in the Details view.

The two portions of the work area on either side of the split bar appear side-by-side. They scroll in unison vertically, but they scroll independently horizontally.

The application is responsible for specifying the position of the split bar, which is defined with the *xVertSplitbar* field. Also, the rightmost column of the left split window is specified with the *pFieldInfoLast* field. *xVertSplitbar* and *pFieldInfoLast* are fields of the CNRINFO data structure.

The left split window cannot be empty if there is data in the right window. The right split window is not required to have data. However, because data cannot be scrolled from the right split window into the left split window, or from left to right, the split bar loses much of its usefulness if the right split window is empty.

The user can drag the vertical split bar within the limits of the window. As the user drags the split bar to the left, the right split window becomes wider; as the user drags the split bar to the right, the left split window becomes wider.

Each container control can have one vertical split bar. Horizontal split bars are not supported.

Using Container Controls

This section provides information about the following topics:

- Creating a container
- Allocating memory for container records
- Allocating memory for container columns
- Inserting container records
- Removing container records
- Setting the container control focus
- Using container views
- Changing a container view
- Arranging icons in a grid

Note: Most of the sample code in this section is part of a complete program that creates a container for a small address book. The program

is illustrated in "Sample Code for Container Controls".

Creating a Container

You create a container by using the WC_CONTAINER window class name in the *ClassName* parameter of WinCreateWindow. Before you create the container, you can create a frame window as a parent. If you create the frame window, it sizes the container to fill its work area. The following sample code shows the code to create both the frame and the container:

```
HAB      hab;
HWND     hPopupMenu;
HWND     hFrameWnd, hCnrWnd;    /* Frame and Container window handles */
PFNWP    SysWndProc;

INT main (VOID)
{
    HMQ      hmq;
    FRAMECDATA fcd;
    QMSG      qmsg;

    if (!(hab = WinInitialize(0)))
        return FALSE;

    if (!(hmq = WinCreateMsgQueue(hab, 0)))
        return FALSE;

    /* *****
    /* Set up the frame control data for the frame window.          */
    /* *****
    fcd.cb = sizeof(FRAMECDATA);
    fcd.flCreateFlags = FCF_TITLEBAR      |
                      FCF_SYSMENU       |
                      FCF_SIZEBORDER    |
                      FCF_SHELLPOSITION |
                      FCF_MINMAX        |
                      FCF_TASKLIST;
    fcd.hmodResources = NULLHANDLE;
    fcd.idResources = 0;

    /* *****
    /* Create the frame to hold the container control.              */
    /* *****
    hFrameWnd = WinCreateWindow(HWND_DESKTOP,
                               WC_FRAME,
                               "Phone Book",
                               0, 0, 0, 0, 0,
                               NULLHANDLE,
                               HWND_TOP,
                               0,
                               &fcd,
                               NULL);

    /* *****
    /* Verify that the frame was created; otherwise, stop.          */
    /* *****
    if (!hFrameWnd)
        return FALSE;

    /* *****
    /* Set an icon for the frame window.                            */
    /* *****
    WinSendMsg(hFrameWnd,
               WM_SETICON,
               (MPARAM)WinQuerySysPointer(HWND_DESKTOP,
                                           SPTR_FOLDER,
                                           FALSE),
               NULL);

    /* *****
    /* Create the container.                                         */
    /* *****
    hCnrWnd = WinCreateWindow(hFrameWnd,
```

```

WC_CONTAINER,
NULL,
CCS_AUTOPOSITION |
CCS_READONLY      |
CCS_SINGLESEL,
0, 0, 0, 0,
hFrameWnd,
HWND_BOTTOM,
FID_CLIENT,
NULL,
NULL);

```

The container is created with a default set of control data, which can be changed using the CM_SETCNRINFO message.

Allocating Memory for Container Records

Your application must allocate memory for a container record by using the CM_ALLOCORECORD message, which also enables you to allocate memory for additional application data.

The maximum number of records is limited by the amount of memory in the user's computer. The container control does not limit the number of records that a container can have.

The following sample code shows how to allocate memory for records that populate the container. A pointer to the record is returned.

```

HWND          hIcon;
PRECORDCORE   Address, FirstRec;
RECORDINSERT  recsIn;
ULONG         x;

/*****
/*  Allocate MAXFRIENDS records all at once -
/*  CM_ALLOCORECORD returns them in a linked list.
*****/
Address = (PRECORDCORE)WinSendMsg(hWnd,
                                CM_ALLOCORECORD,
                                0,
                                MPFROMLONG(MAXFRIENDS));

```

Your application can use the CM_ALLOCORECORD message to allocate memory for one or more container records. The application can request *n* container records with an *nRecords* parameter. If *n* is one, a pointer to that record is returned. If *n* is greater than one, a pointer to the first record in a linked list of *n* records is returned.

Allocating Memory for Container Columns

In addition to allocating memory for records, an application also must allocate memory for columns of data if the details view is used. In the Details view, a container's data is displayed in columns, each of which is described in a FIELDINFO data structure.

Memory is allocated for FIELDINFO data structures using the CM_ALLOCODETAILFIELDINFO message. Unlike the CM_ALLOCORECORD message, the CM_ALLOCODETAILFIELDINFO message does not allow the application to allocate memory for additional application data. However, the *pUserData* field of the FIELDINFO data structure can be used to store a pointer to the application-allocated data.

Multiple FIELDINFO data structures can be allocated with the *nFieldInfo* parameter of the CM_ALLOCODETAILFIELDINFO message.

Inserting Container Records

After the memory is allocated, you can insert one or more container records by using the CM_INSERTRECORD message.

The following sample code inserts records into a container for which memory was allocated in the previous code fragment:

```

/*****
/* We will need the first record's address to
/* insert them into the container.
*****/
FirstRec = Address;

/*****
/* Loop through the address book, loading as we go.
/* Because the CM_ALLOCORECORD returns a linked
/* list, the address of the next record is retrieved
/* from each record as we go (preccNextRecord).
*****/
for (x = 0; x < MAXFRIENDS; x++)
{
    Address->cb = sizeof(RECORDCORE); /* Standard records
    Address->hptrIcon = hIcon; /* File icon
    Address->pszIcon = Friends[x].NickName;
    Address->pszName = Friends[x].FullName;
    Address->pszText = Friends[x].FullName;
    Address = Address->preccNextRecord; /* Next record in list
}

/*****
/* Set up the insert record structure to place the
/* records in the container.
*****/
recsIn.cb = sizeof(RECORDINSERT);

/* Put the records in after any others */
recsIn.pRecordOrder = (RECORDCORE)CMA_END;

/* All the records are top level (not children of other records) */
recsIn.pRecordParent = NULL;

/* The icons are top level */
recsIn.zOrder = (USHORT)CMA_TOP;

/* Redraw the container */
recsIn.fInvalidateRecord = TRUE;

/* Set the number of records to insert */
recsIn.cRecordsInsert = MAXFRIENDS;

/*****
/* Insert the records into the container.
*****/
WinSendMsg(hWnd,
            CM_INSERTRECORD,
            (RECORDCORE)FirstRec,
            &recsIn);
}

```

The CM_INSERTRECORD message requires you to provide two pointers. The first pointer points to the record that is to be inserted, which is specified in the *FirstRec* parameter. When you are inserting multiple records, use this parameter to specify a pointer to a linked list of records.

The second pointer points to a RECORDINSERT data structure (*&recsIn*), which specifies information the container needs for inserting records.

One of the elements of information that this data structure contains is the order in which the records are to be inserted, which is specified in the *pRecordOrder* field. In this field you have two options. The first option is to specify a pointer to a container record. The records being inserted are placed immediately after that record. In this case, the *pRecordParent* field is ignored.

The second option is to specify whether the records being inserted are to be placed at the beginning or end of a list of records. This is done by specifying either the CMA_FIRST or CMA_END attributes. If you choose this option, the list of records used depends on the value of the *pRecordParent* field.

If CMA_FIRST or CMA_END is specified and the value of the *pRecordParent* field is NULL, the inserted records are placed at the beginning or end of the root-level records. However, if CMA_FIRST or CMA_END is specified and *pRecordParent* contains a pointer to a parent item

record, the records are inserted at the beginning or end of the list of child item records that this parent record contains.

The RECORDINSERT data structure also lets you specify the z-order position of the records being inserted. The CMA_TOP and CMA_BOTTOM attributes of the *zOrder* field place the record at the top or bottom, relative to the other records in the z-order list. This field applies to the Icon view only.

To specify the number of records that are being inserted, use the *cRecordsInsert* field. The value of this field must be greater than 0.

The last field in the RECORDINSERT data structure is *fillInvalidateRecord*, which enables you to control whether the records are displayed automatically when they are inserted. If you specify TRUE in this field, the display is updated automatically. However, if you specify FALSE, the application must send the CM_INVALIDATERECORD message after the records are inserted to update the display.

Where items are positioned in a container depends on the view the user has specified. If the Icon view is specified and the CCS_AUTOPOSITION style bit is not set, the *x*- and *y*-coordinates for each record, which are stored in the *ptIcon* field of the RECORDCORE and MINIRECORDCORE data structures, determine its position. Records displayed in the Name view, Text view, Tree view, and Details view are positioned as previously described in this section.

Note: Records inserted into a list of child record items can be displayed in the Tree view only. These records are visible only if the parent record item to which these child record items belong is expanded.

Removing Container Records

The CM_REMOVERECORD message can be used to remove one or more container records from the container control. The following sample code removes all records from a container and frees the memory associated with those records. It is the application's responsibility to free all application-allocated memory that is associated with the removed container records. The container is invalidated and repainted.

```
USHORT cNumRecord;           /* Number of records to be removed */
USHORT fRemoveRecord;        /* Container message attributes */

/*****
/* Zero means remove all records.
*****/

cNumRecord = 0;

/*****
/* Specify attributes to invalidate the container
/* and free the memory.
*****/
fRemoveRecord =
    CMA_INVALIDATERECORD | CMA_FREE;

/*****
/* Remove the records.
*****/
WinSendMsg(hwndCnr,
    CM_REMOVERECORD,
    NULL,
    MPFROM2SHORT(
        cNumRecord,
        fRemoveRecord));
```

The application must set the pointers to each record to be removed in an array. If the *fRemoveRecord* parameter of this message includes the CMA_FREE attribute, the records are removed and the memory is freed. If this attribute is not set, the records are removed from the list of items in the container, and the application must use the CM_FREERECORD message to free the memory. The default is not to free the memory.

If the *fRemoveRecord* parameter includes the CMA_INVALIDATERECORD attribute, the container is invalidated after the records are removed. The default is not to invalidate the container. The CMA_INVALIDATERECORD attribute can be used with the CMA_FREE attribute, separated by a logical OR operator (|), to free the record's memory and invalidate the container.

Setting the Container Control Focus

The application must set the focus to the container control using `WinSetFocus` so that all mouse and keyboard activity goes to the container window. The following code fragment shows how to use `WinSetFocus`:

```
WinSetFocus(HWND_DESKTOP,      /* Desktop window handle */
            hListWnd)          /* Handle of window to receive focus */
```

Using Container Views

Container views are specified by using attributes on the *flWindowAttr* field of the `CNRINFO` data structure.

Because the container control does not support both icons and bit maps in the same view, an application must specify which are used by setting either the `CA_DRAWICON` attribute or the `CA_DRAWBITMAP` attribute in the *flWindowAttr* field of the `CNRINFO` data structure. The default is the `CA_DRAWICON` attribute. The size of the icon or bit map can be specified in the *slBitmapOrIcon* field of the `CNRINFO` data structure.

Changing a Container View

The following sample code shows how to use the `CM_SETCNRINFO` message to change from the current view of a container (Name, Details, or Text) to the Icon view:

```
CNRINFO cnrInfo;

/*****
 * Set the attribute field to the Icon view.
 *****/
cnrInfo.flWindowAttr = CV_ICON;

/*****
 * Change the view from the current view to the Icon view.
 *****/
WinSendMsg(
    hwndCnr,          /* Container window handle */
    CM_SETCNRINFO,     /* Container message for setting */
    MPFROMP(&cnrInfo), /* Container control data */
    MPFROMLONG(
        CMA_FLWINDOWATTR)); /* Message attribute that sets
                             /* container window attributes
```

Creating a Grid for Icons

This section will describe how to create a grid.

Graphical User Interface Support for Container Controls

This section describes the container control support for graphical user interfaces (GUIs). Except where noted, this support conforms to CUA interface design guidelines. The GUI support provided by the container control consists of the following:

- Scrolling
- Dynamic scrolling
- Selecting container items
- Providing emphasis
- Using direct manipulation

Scrolling

The container control automatically provides horizontal or vertical scroll bars, or both, whenever the container window's work area is not large enough to display all of the container items.

If all container items are visible in the work area, the scroll bars are either removed or disabled, depending on the view and how the items are positioned, as follows:

- If container items are displayed in the icon or tree view, and one or more items are not visible in the work area, a horizontal scroll bar, vertical scroll bar, or both, are provided, depending on the position of the items outside of the work area. If container items are positioned to the right or left of the work area, a horizontal scroll bar is provided; if container items are positioned below or above the work area, a vertical scroll bar is provided.

Scroll bars are not provided if all the container items are visible in the work area. Scroll bars are removed from the container window if either of the following occurs:

- Container items positioned outside the work area are moved into the work area.
- The size of the container window is increased so that container items formerly not visible become visible.
- If container items are displayed in non-flowed text and non-flowed Name views, a vertical scroll bar is provided; this scroll bar is disabled if all the container items are visible in the work area. A horizontal scroll bar is used in these views only when the work area is too narrow to allow the widest container item to be seen in its entirety. If the user changes the window size to allow the entire widest container item to be seen, the horizontal scroll bar is removed.
- If container items are displayed in flowed text and flowed name views, a horizontal scroll bar is provided; this scroll bar is disabled if all the container items are visible in the work area. A vertical scroll bar is used in these views only when the work area is too short to allow the tallest container item to be seen in its entirety. If the user changes the window size to allow the entire tallest container item to be seen, the vertical scroll bar is removed.
- If container items are displayed in the Details view, both horizontal and vertical scroll bars are provided. These scroll bars are disabled if all the container items are visible in the work area.

Note: A Details view that is split has two horizontal scroll bars, one for each portion of the split window.

Dynamic Scrolling

The container control supports *dynamic scrolling*, which enables the user to drag the scroll box in the scroll bar and get immediate visible feedback on where the scrolling stops when the scroll box is dropped. If the scrolling range is greater than 32K pels, dynamic scrolling is disabled.

Selecting Container Items

Except during direct manipulation and direct editing of text in a container, a user must select a container item before performing an action on it. The container control provides several selection types, along with selection techniques to implement those types. The container control also supports two selection mechanisms: pointing device, such as a mouse, and the keyboard.

Selection Types

The container control supports the following selection types:

- **Single selection**

Single selection enables a user to select only one container item at a time. This is the default selection type for all views and is the only selection type supported for the Tree view.
- **Extended selection**

Extended selection enables a user to select one or more container items, in any combination. The CUA-defined keyboard augmentation keys are implemented for extended selection. When used with a pointing device, these keys enable a user to select discontinuous sets of container items. Extended selection is valid for all views except the Tree view.
- **Multiple selection**

Multiple selection enables a user to select none, some, or all of the container items. Multiple selection is valid for all views except the Tree view.

Only one of these selection types can be used for each container. The selection type for a container is defined when the container is created.

Selection Techniques

Depending on the type of view and the type of selection, a user can select container items using the following selection techniques:

- **Marquee selection**

Marquee selection is supported only in the Icon view and is only valid with the extended and multiple selection types. This selection technique enables a user to begin selection from an anchor point that is established by moving the pointer to white space in the container and pressing, but not releasing, the select button on the pointing device. As the user presses the select button and drags the pointer, a tracking rectangle is drawn between the anchor point and the current pointer position. All items whose icons or bit maps are entirely within the tracking rectangle are dynamically selected.
- **Swipe selection**

Swipe selection is valid only with the extended and multiple selection types. The container control implements two techniques for swipe selection: touch swipe and range swipe.
 - **Touch swipe**

Touch-swipe selection is implemented in the Icon view. With this selection technique, the pointer must pass over some portion of a container item while the user is pressing the select button for that item to be selected.
 - **Range swipe**

In views other than the Icon and Tree views, range-swipe selection is available. With this method, the user presses the select button while moving the pointer. However, the pointer does not have to pass directly over a container item for that item to be selected. Aside from pressing the select button and moving the pointer, the only other requirement for selection is that the container item must be within a range of items that is being selected. The range begins at the pointer's position when the user presses the select button; it ends at the pointer's position when the user releases the select button.
- **First-letter selection**

For the Icon, Name, Text, and Tree views, first letter selection occurs when a character key is pressed, and the first container item whose text begins with that character is displayed with selected-state emphasis. The same is true for the Details view, except that all the columns for a record are searched for a matching character before the next record is searched. The effect of first letter selection on other selected container items depends on the chosen selection type (single, multiple, or extended).

Note: If more than one container window is open, selecting a container item in one window has no effect on the selections in any other window.

Selection Mechanisms

Mouse button 1 (the select button) is used for selecting container items, and mouse button 2 (the drag button) is used for dragging and dropping container items during direct manipulation. These definitions also apply to the same buttons on any other pointing device.

In addition, a user can press a keyboard key while pressing a mouse button; this is called *keyboard augmentation*. The only instance of keyboard augmentation defined specifically for the container control is pressing the Alt key with the select button, which starts direct editing of text in a container.

In addition, the container control supports two keyboard cursors that can be moved by using keyboard navigation keys:

- The *selection cursor*, a dotted black box drawn around a container item, which represents the current position for the purpose of keyboard navigation.
- The *text cursor*, a vertical line that shows the user where text can be inserted or deleted when container text is being edited directly.

Keyboard navigation consists of the use of the Up and Down Arrow, Left and Right Arrow, Home, End, PgUp, and PgDn keys. If container items are not visible within the work area, navigation with these keys causes the items to scroll into view if the user is not editing container text directly.

Providing Emphasis

The container control supports various types of emphasis. The following list describes forms of emphasis that have a distinct visible representation in the container control:

- Selected-state and unavailable-state emphasis

When a container item is selected, the container item receives *selected-state emphasis*, which means that the emphasis is applied to icon/text or bit-map/text pairs in the Icon, Name, Tree icon, and Tree name views; text strings in the Text and Tree text views; and an entire row that represents a container item in the Details view.

The color for selected-state emphasis can be changed by using the control panel or WinSetPresParam, which results in a WM_PRESPARAMCHANGED message being sent to the container.

- In-use emphasis

Cross-hatching behind an icon or bit map indicates *in-use emphasis*. In-use emphasis is not applied to container items in the Text view, Tree text view, or Details view when it contains text only. However, the Details view often includes icons or bit maps in one column of each record, usually the leftmost column. In this situation, specify the column that contains the icons or bit maps so that in-use emphasis can be applied to them. This column can be set by using the *pFieldInfoObject* field of the CNRINFO data structure.

- Target emphasis

Target emphasis is used during direct manipulation. When a user drags one container item over another, the item beneath the dragged item displays *target emphasis*. Two forms of target emphasis (visible feedback) are available: a black line and a black border. These forms of emphasis indicate the *target*, where the container item is dropped if the user releases the drag button.

Using Direct Manipulation

Direct manipulation is a protocol that enables the user to drag a container item within its current window or from one window to another. The user can drop the container item either on white space in a window or on another item.

Direct manipulation can be performed with all views of the container control. A function is provided so that the application is notified if an item is dropped on another item in the container and if an item is dragged from the container.

The user can drag any container item, whether or not it is selected. If the user presses the drag button when the pointer is over a selected container item, the application drags all selected items.

If the user presses the drag button when the pointer is over a container item that is not selected, the application drags only the item that the pointer is over.

The container control fully supports direct manipulation.

Enhancing Container Controls Performance and Effectiveness

The following topics offer information about fine-tuning a container to enhance its performance and effectiveness:

- Positioning container items
 - Specifying space between container items
 - Providing target emphasis
 - Specifying deltas for large amounts of data
 - Direct editing of text in a container
 - Specifying container titles
 - Specifying fonts and colors
 - Drawing container items and painting backgrounds
 - Filtering container items
 - Optimizing container memory usage
 - Sharing records among multiple containers
-

Positioning Container Items

Container items are positioned in the Icon view according to workspace coordinates.

The *workspace* is a two-dimensional Cartesian-coordinate system. The user can see a portion of the workspace in the work area, which is the scrollable viewing area of the container that is defined by the size of the container window. The work area is logically scrollable within the workspace.

Scrollable Workspace Areas

The workspace is indicated by the solid black line that runs even with:

- The top and bottom edges of the topmost and bottommost container items
- The left and right edges of the leftmost and rightmost container items

The workspace is defined by the minimum and maximum x - and y -coordinates of the items in the container. The work area of the container window can be scrolled only within the workspace and only as far as is necessary to see the topmost, bottommost, leftmost, and rightmost container items.

Workspace and Work Area Origins

When the container is created, the work area and workspace share the same origin, (0,0). If the application requires that the work area and the workspace have different origins, the application can use the *ptlOrigin* field of the CNRINFO data structure and the CM_SETCNRINFO message to set the origin of the work area. The application can use the CM_QUERYCNRINFO and CM_SETCNRINFO messages to obtain the origin when the user ends the application and to reset the origins when the user restarts the application.

Container items are located in reference to the workspace origin. There is a visual shift as the work area is scrolled; however, because the work area moves over a fixed workspace, the coordinates of the container items do not change.

Specifying Space between Container Items

You can specify the amount of vertical space, in pels, to allow between container items by using the *cyLineSpacing* field of the CNRINFO data structure. If you do not specify how much vertical space can be used, the container control sets the space between the items using a default value. For the Tree view, you can specify the horizontal distance between the levels by using the *cxTreeIndent* field of the CNRINFO data structure. If this value is less than 0, a default is used.

Providing Source Emphasis

Source emphasis is the type of emphasis provided when a context menu is displayed. It appears as a dotted box with rounded corners that surrounds the item for which the context menu is requested, or the item that is being dragged.

To provide source emphasis for container items issue the CM_SETRECORDEMPHASIS message with the following values:

1. Provide a pointer to the RECORDCORE or MINIRECORDCORE data structure in *param1*.
You can provide source emphasis for the entire container by setting *param1* to NULL.
2. Set the *usChangeEmphasis* parameter to TRUE in *param2*.
3. Set the *fEmphasisAttribute* parameter in *param2* to CRA_SOURCE (0x00004000L).

To remove source emphasis follow the same procedure outlined above, but set the *usChangeEmphasis* parameter in *param2* to FALSE instead of TRUE.

Providing Target Emphasis

The CA_ORDEREDTARGETEMPH and CA_MIXEDTARGETEMPH attributes of the CNRINFO data structure's *fWindowAttr* field determine the form of emphasis applied for the Text, Name, and Details views, as follows:

- If the CA_ORDEREDTARGETEMPH attribute is set:
 - The CN_DRAGAFTER notification code is sent when a container item is being dragged.
 - A black line is drawn between container items to show the current target position.
- If the CA_MIXEDTARGETEMPH attribute is set:

- The CN_DRAGAFter and CN_DRAGOVER notification codes are sent when a container item is being dragged. The notification code sent depends on the position of the pointer relative to the item it is positioned over.
- A black line is drawn if the pointer is positioned such that the item being dragged is inserted between two target items.
- A black border is drawn around either the entire target item for the Text and Details views or the icon or bit map for the Name view if the pointer is positioned such that the item being dragged is dropped on the target item.
- If the CA_ORDEREDTARGETEMPH and CA_MIXEDTARGETEMPH attributes are not set:
 - The CN_DRAGOVER notification code is sent when a container item is being dragged.
 - A black border is drawn around the entire target item for the Text and Details views, and around the icon or bit map only for the Name view.

For the Icon and Tree view, the CA_ORDEREDTARGETEMPH and CA_MIXEDTARGETEMPH attributes are ignored, so target emphasis is applied as follows:

- The CN_DRAGOVER notification code is sent when a container item is dragged.
- A black border is drawn around the target, as follows:
 - For the Icon view, if the target is another container item, a black border is drawn around the icon or bit map that represents the container item, but not around the text string beneath it. If the target is white space, a black border is drawn around the outer edge of the entire work area.
 - For the Tree icon and Tree name views, a black border is drawn around the icon or bit map that represents the container item, but not around the text string to the right of it.
 - For the Tree text view, a black border is drawn around the entire target item.

Specifying Deltas for Large Amounts of Data

The container control can accommodate large amounts of data with an application-defined delta. The *delta* is an application-defined threshold, or number of container items, from either end of the list. The application is responsible for specifying the delta value in the CNRINFO data structure's *cDelta* field. It also is responsible for setting the delta value with the CMA_DELTA attribute of the CM_SETCNRINFO message's *ulCnrInfoF* parameter.

The container control monitors its place in the list of container items when the user is scrolling through it. When the user scrolls to the delta from either end of the list, the container control sends a CN_QUERYDELTA notification code to the application as a request for more container items in the list.

The application is responsible for managing the records in the container. When the application receives the CN_QUERYDELTA notification code, the application is responsible for removing and inserting container records by using the CM_REMOVERECORD message and the CM_INSERTRECORD message, respectively.

Note:

The delta concept is intended for applications with large amounts of data, or several thousand records. Applications with smaller amounts of data are not required to use the delta function. The default delta value is 0.

The delta function is not available in the Icon view because it is intended for data displayed in a linear format.

Direct Editing of Text in a Container

Direct editing of text is supported for any text field in a container, including the container title, column headings, and container items. If a text field, such as the text field beneath an icon in the Icon view, has no text and is not read-only, a user can place text in that field by editing the field directly. The font specified for the container by the application is used for the edited text.

Direct editing is supported only for text data. Therefore, if the data type in the Details view is other than CFA_STRING, a user cannot edit it. CFA_STRING is an attribute of the FIELDINFO data structure's *//Data* field.

You can prevent a user from editing any of the text in a container window by setting the CCS_READONLY style bit when a container is created. If you do not set this style bit, the user can edit any of the text in a container window unless you set the following read-only attributes:

- CA_TITLEREADONLY of the CNRINFO data structure's *//WindowAttr* field
- CRA_RECORDREADONLY of the RECORDCORE data structure's *//RecordAttr* field
- CFA_FIREADONLY of the FIELDINFO data structure's *//Data* field
- CFA_FITITLEREADONLY of the FIELDINFO data structure's *//Title* field

If one of these read-only attributes is set, a user's attempts to edit container text directly are ignored.

A user can edit container text directly by doing either of the following:

- Moving the pointer to an editable text field, holding down the Alt key, and clicking the select button
- Sending a CM_OPENEDIT message to the container control. The application can assign a key or menu choice to this message so that the keyboard can be used to edit container text directly.

The container control responds by using the WM_CONTROL message to send the CN_BEGINEDIT notification code to the application. A window that contains a multiple-line entry (MLE) field opens to show that container text can be edited directly.

The editing actions supported by MLEs, such as **Cut**, **Copy**, and **Paste**, are also supported by the container control. These actions can be performed using system-defined *shortcut* keys. The actions and shortcut keys are defined by CUA interface design guidelines.

If the user enters a text string that is longer than the text field, the text string scrolls. If multiple lines of text are needed or wanted, a user can press the Enter key to insert a new line.

A user can end the direct editing of container text and save the changes by doing either of the following:

- Moving the pointer outside the MLE and pressing the select button
- Sending a CM_CLOSEEDIT message to the container control. The application can assign a key or menu choice to this message so that the keyboard can be used to end the direct editing of container text.

The container responds by sending the WM_CONTROL message to the application again, but this time with the CN_REALLOCPSZ notification code. The application can allocate more memory on receipt of the CN_REALLOCPSZ notification code, if necessary. If the application returns TRUE, the container control copies the new text to the application's text string. If the application returns FALSE, the text change in the MLE is disregarded. The container then sends the WM_CONTROL message to the application again, this time with the CN_ENDEDIT notification code. The MLE field is removed from the screen, leaving only the text string.

A user can end the direct editing of container text without saving any changes to the text in numerous ways, including the following:

- Pressing the Esc key
- Dragging the container item that is being edited
- Pressing the Alt key and the select button before the direct editing of container text has ended
- Scrolling the container window

The CN_ENDEDIT notification code is sent to the application in each of these cases.

Searching for Exact Text String Matches

There might be times when you need to search the container for a text string that is an exact match of your search string argument. To find an exact match:

- In the SEARCHSTRING data structure, specify values for the fields as you normally would, with the following exception:

Along with an attribute for the type of view being displayed in the container, in the *usView* field specify the CV_EXACTLENGTH (0x10000000L) flag. For example:

```
CV_EXACTLENGTH | CV_ICON
```

Note: The *usView* field is used for specifying the exact match attribute, and the type of view. Despite the "us" prefix this field is a ULONG. The "us" prefix is used in the header files to maintain backward compatibility.

Specifying Container Titles

The container control can have a non-scrollable title that consists of one or more lines of text. The container control does not limit the number of lines or the number of characters in each line. If specified, this title is the first line or lines of the container control. The text of the title is determined by the application and can be used to identify the container or to contain status information.

The CA_CONTAINERTITLE attribute must be set to include a title in a container window. The default is no container title.

If you do not want the user to be able to edit the container title directly, you can set the CA_TITLEREADONLY attribute. The default is that the container title can be edited.

Below the title a horizontal line separates the container title from the container items. The CA_TITLESEPARATOR attribute must be set in order to include a separator line in a container window. The default is no separator line.

The container titles in both figures are centered. This is the default. However, the CA_TITLECENTER, CA_TITLELEFT, or CA_TITLERIGHT attribute can be used to specify whether a container title is to be centered, left-justified, or right-justified.

All the container attributes described here are attributes of the CNRINFO data structure's *fWindowAttr* field.

Specifying Fonts and Colors

A different font can be specified for each view. The same font is used for the text within each view. Text color can be configured from the system control panel. The application can override the system-defined font and colors by using WinSetPresParam.

The font and color can be changed for the text in all views. However, font and color cannot be changed for text in individual columns in the Details view. Therefore, all text in the details view, including the container title, columns, and column headings, has the same font and color.

Drawing Container Items and Painting Backgrounds

The container control enables your application to paint the container's background, draw the container items, or both. If the CA_OWNERPAINTBACKGROUND attribute is set, the container control sends the CM_PAINTBACKGROUND message to itself. Your application can control background painting by subclassing the container control and intercepting the CM_PAINTBACKGROUND message. CA_OWNERPAINTBACKGROUND is an attribute of the CNRINFO data structure's *fWindowAttr* field.

To support *ownerdraw*, the drawing of container items by the application, the container control provides the CA_OWNERDRAW attribute of the CNRINFO data structure's *fWindowAttr* field. If this attribute is set and the application processes the WM_DRAWITEM window message, the application is responsible for drawing each container item, including the types of emphasis.

In addition, the container control supports ownerdraw for each column in the Details view. This support is indicated by the CFA_OWNER attribute, which is specified in the FIELDINFO data structure's *fData* field.

If the CA_OWNERDRAW attribute or CFA_OWNER attribute is set, the container control sends the application a WM_DRAWITEM message with a pointer to an OWNERITEM data structure as the *owneritem* parameter.

Filtering Container Items

If the CRA_FILTERED attribute is set for a container item, that item is not displayed. Therefore, filtering can be used to hide container items.

CRA_FILTERED is an attribute of the RECORDCORE data structure's *//RecordAttr* field.

Optimizing Container Memory Usage

The container control provides an option to enable you to develop applications that minimize the amount of memory used for each container record. This is done by specifying the CCS_MINIRECORDCORE style bit when the container is created, which causes a smaller version of the RECORDCORE data structure, MINIRECORDCORE, to be used. The following table shows the differences between these two data structures:

RECORDCORE	MINIRECORDCORE
Up to eight image handles can be specified for each record.	Only one image handle can be specified for each record. Note: This image must be an icon.
Up to four text strings can be specified for each record.	Only one text string can be specified for each record.

Allocating Memory for when Using MINIRECORDCORE

The following sample code shows how to allocate memory for one container record when the MINIRECORDCORE data structure is used. A pointer to the MINIRECORDCORE data structure is returned.

```
HWND          hwndCnr;          /* Container window handle          */
PMINIRECORDCORE pRecord;        /* Pointer to MINIRECORDCORE structure */
ULONG          nRecords = 1;    /* 1 record to be allocated          */

pRecord =
(PMINIRECORDCORE)WinSendMsg(
    hwndCnr,          /* Container window handle          */
    CM_ALLOCORECORD,  /* Message for allocating the record */
    NULL,             /* No additional memory              */
    (MPARAM)nRecords); /* Number of records to be allocated */
```

Sharing Records among Multiple Containers

The container control enables the application to share records that are allocated among multiple containers in the same process. That is, records can be allocated once and then inserted into many containers in the same process. Only one copy of each record is in memory, but the container provides the flexibility for the records to appear as though they are independent of one another.

When a record is inserted into the container, the *//RecordAttr* and *ptIcon* fields of the record structure are saved internally. The values in these fields cause the record attributes for all views and the icon position for the Icon view to be associated with the specific container into which the record is inserted. If the same record is inserted into multiple containers, the attributes and icon location of each record are maintained separately. The application uses the CM_QUERYRECORDINFO message to retrieve the current values of these two fields for a particular record in a specific container.

Invalidating Records Shared by Multiple Containers

When a record is invalidated by an application, the *flRecordAttr* and *ptllcon* fields are saved internally, just as when a record is inserted. The CM_QUERYRECORDINFO message is used to acquire the current data for each record that is being invalidated. After querying the current data, the data can be changed before invalidating its record.

Freeing Records Shared by Multiple Containers

When an application attempts to free a record in an open container, the record is freed only if it is not being used in any other open container. The methods of freeing records in an open container are to use the CM_FREERECORD message, or use the CM_REMOVEVERECORD message and specify the CMA_FREE attribute.

Sample Code for Container Controls

This section illustrates a complete container control sample program. Several parts of this program are explained in "Using Container Controls".

Container Application Sample Code

The container application includes the following files:

- Contain.C
- Contain.RC
- Contain.H
- Contain.LNK
- Phones.H

The following sample illustrates the container application code:

```
=====
CONTAIN.C
=====

#define INCL_WIN
#define INCL_GPI

#include <os2.h>
#include <stdio.h>
#include <string.h>
#include "contain.h"
#include "phones.h"

/*****
/*  Program Overview:
/*
/*  This program creates a frame window as a parent, then creates
/*  a container window as a child. The frame window sizes the
/*  container to fill its client area.
/*
/*
/*  After the windows are created successfully, the container
/*  window is populated. First, the container is sent a message to
/*  allocate memory for each of the records which will be inserted.
/*  After the memory is allocated, we set the values for each record.
/*  (This sample program reads data from a static array - you could
/*  also load values from a file.) Then, the container is sent a
/*  message to insert the records (which makes them visible).
*****/
```

```

/*
/* This container is read-only, which means the end user cannot
/* change the title text. It supports single selection.
/*
/* In the message loop, we must check for WM_CONTROL messages,
/* which are generated from the container control. This sample
/* processes CN_ENTER messages, when an item in the container is
/* selected (either with the mouse or the keyboard), and
/* CN_CONTEXTMENU messages, when a context menu is requested. The
/* context menu allows the user to change the display mode of the
/* container. Our container supports Icon, Text, and Name views.
/*
/* When a CN_ENTER message is received, we loop through the array
/* of names until we find a match. On a match, we pop up a message
/* box which contains the nickname, name, and number of the person
/* selected.
/*
/*****
#pragma linkage (main,optlink)
INT main(VOID);
VOID LoadDatabase(HWND);

/*****
/* Main() - program entry point.
/*****
MRESULT EXPENTRY LocalWndProc(HWND, ULONG, MPARAM, MPARAM);

HAB      hab;
HWND     hPopupMenu;
HWND     hFrameWnd, hCnrWnd;
PFNWP    SysWndProc;

INT main (VOID)
{
    HMQ      hmq;
    FRAMECDATA fcd;
    QMSG     qmsg;

    if (!(hab = WinInitialize(0)))
        return FALSE;

    if (!(hmq = WinCreateMsgQueue(hab, 0)))
        return FALSE;

/*****
/* Set up the frame control data for the frame window.
/*****
fcd.cb = sizeof(FRAMECDATA);
fcd.flCreateFlags = FCF_TITLEBAR |
                  FCF_SYSMENU |
                  FCF_SIZEBORDER |
                  FCF_SHELLPOSITION |
                  FCF_MINMAX |
                  FCF_TASKLIST;
fcd.hmodResources = NULLHANDLE;
fcd.idResources = 0;

/*****
/* Create the frame to hold the container control.
/*****
hFrameWnd = WinCreateWindow(HWND_DESKTOP,
                          WC_FRAME,
                          "Phone Book",
                          0, 0, 0, 0, 0,
                          NULLHANDLE,
                          HWND_TOP,
                          0,
                          &fcd,
                          NULL);

/*****
/* Verify that the frame was created; otherwise, stop.
/*****
if (!hFrameWnd)
    return FALSE;

/*****
/* Set an icon for the frame window.
/*****

```

```

/*****
WinSendMessage(hFrameWnd,
               WM_SETICON,
               (LPARAM)WinQuerySysPointer(HWND_DESKTOP,
                                           SPTR_FOLDER,
                                           FALSE),
               NULL);

/*****
/* Create the container.
/*****
hCnrWnd = WinCreateWindow(hFrameWnd,
                          WC_CONTAINER,
                          NULL,
                          CCS_AUTOPOSITION |
                          CCS_READONLY |
                          CCS_SINGLESEL,
                          0, 0, 0, 0,
                          hFrameWnd,
                          HWND_BOTTOM,
                          FID_CLIENT,
                          NULL,
                          NULL);

/*****
/* If we got it, fill it up.
/*****
if (hCnrWnd)
    LoadDatabase(hCnrWnd);

/*****
/* We must intercept the frame window's messages
/* (to capture any input from the container control).
/* We save the return value (the current WndProc),
/* so we can pass it all the other
/* messages the frame gets.
/*****
SysWndProc = WinSubclassWindow(hFrameWnd, (PFNWP)LocalWndProc);

/*****
/* Load the popup menu from the resources
/* and show the frame window.
/*****
hPopupMenu = WinLoadMenu(HWND_OBJECT, NULLHANDLE, IDM_DISPLAY);

WinShowWindow(hFrameWnd, TRUE);

/*****
/* Standard PM message loop - get it, dispatch it.
/*****
while (WinGetMsg(hab, &qmsg, NULLHANDLE, 0, 0))
{
    WinDispatchMsg(hab, &qmsg);
}

/*****
/* Clean up on the way out.
/*****
WinDestroyMsgQueue(hmq);
WinTerminate(hab);

return TRUE;
}

/*****
/* LocalWndProc() - window procedure for the frame window.
/* Called by PM whenever a message is sent to the frame.
/*****
HRESULT EXPENTRY LocalWndProc(HWND hwnd, ULONG msg, LPARAM mp1, LPARAM mp2)
{
    char                szBuffer[80];
    CNRINFO             cnrInfo;
    PNOTIFYRECORDENTER Selected;
    POINTL              pt;
    int                 xi;

    switch(msg)
    {
        case WM_CONTROL:

```

```

switch (SHORT2FROMMP(mp1))
{
/*****
/* Context menu - usually right mouse button clicked
/* on window. Popup a menu to allow the user to
/* select a new view of the container.
*****/
case CN_CONTEXTMENU:
    WinQueryPointerPos(HWND_DESKTOP, &pt);
    WinPopupMenu(HWND_DESKTOP,
        hwnd,
        hPopupMenu,
        (SHORT)pt.x,
        (SHORT)pt.y,
        IDM_ICON,
        PU_NONE |
        PU_MOUSEBUTTON1 |
        PU_KEYBOARD |
        PU_SELECTITEM);

    break;

case CN_ENTER:
/*****
/* User selected an item - we take the icon text
/* and spin through the array of Friends, looking for
/* a match - on match, print out the phone number
*****/
    Selected = (PNOTIFYRECORDENTER)mp2;
    for (x = 0; x < MAXFRIENDS; x++)
    {
        if (!strcmpi(Friends[x].NickName,
            Selected->pRecord->pszIcon))
        {
            sprintf(szBuffer,
                "'%s' (%s) %s",
                Friends[x].NickName,
                Friends[x].FullName,
                Friends[x].Phone);
            WinMessageBox(HWND_DESKTOP,
                HWND_DESKTOP,
                szBuffer,
                "Phone",
                0,
                MB_OK);
        }
    }
    break;
}
break;

case WM_COMMAND:
    switch (SHORT1FROMMP(mp1))
    {
        case IDM_ICON:
            cnrInfo.flWindowAttr = CV_ICON;
            break;
        case IDM_NAME:
            cnrInfo.flWindowAttr = CV_NAME;
            break;
        case IDM_TEXT:
            cnrInfo.flWindowAttr = CV_TEXT;
            break;
        default:
            return (*SysWndProc)(hwnd, msg, mp1, mp2);
            break;
    }

    WinSendMsg(hCnrWnd,
        CM_SETCNRINFO,
        &cnrInfo,
        MPFROMLONG(CMA_FLWINDOWATTR));

    break;

/*****
/* Send the message to the usual WC_FRAME WndProc.
*****/

```

```

        default:
            return (*SysWndProc)(hwnd, msg, mp1, mp2);
            break;
    }
    return (*SysWndProc)(hwnd, msg, mp1, mp2);
}

/*****
/* LoadDatabase() - utility function
/* called after the WC_CONTAINER window is created successfully,
/* allocates and populates container records, and then inserts
/* the records into the container window.
*****/
VOID LoadDatabase (HWND hwnd)
{
    HWND          hIcon;
    RECORDCORE    Address, FirstRec;
    RECORDINSERT  recsIn;
    ULONG         x;

/*****
/* The Icon view for each of the records in the
/* container will use the standard File icon,
/* so we grab the handle now for reference later.
*****/
    hIcon = WinQuerySysPointer(HWND_DESKTOP, SPTR_FILE, FALSE);

/*****
/* Allocate MAXFRIENDS records all at once -
/* CM_ALLOCORECORD returns them in a linked list.
*****/
    Address = (RECORDCORE)WinSendMsg(hwnd,
                                     CM_ALLOCORECORD,
                                     0,
                                     MPFROMLONG(MAXFRIENDS));

/*****
/* We will need the first record's address to
/* insert them into the container.
*****/
    FirstRec = Address;

/*****
/* Loop through the address book, loading as we go.
/* Because the CM_ALLOCORECORD returns a linked list,
/* the address of the next record is retrieved
/* from each record as we go (preccNextRecord).
*****/
    for (x = 0; x < MAXFRIENDS; x++)
    {
        Address->cb          = sizeof(RECORDCORE); /* Standard records
        Address->hptrIcon    = hIcon;              /* File icon
        Address->pszIcon     = Friends[x].NickName;
        Address->pszName     = Friends[x].FullName;
        Address->pszText     = Friends[x].FullName;
        Address = Address->preccNextRecord; /* Next record in list
    }

/*****
/* Set up the insert record structure to place
/* the records in the container.
*****/
    recsIn.cb = sizeof(RECORDINSERT);

    /* Put the records in after any others */
    recsIn.pRecordOrder = (RECORDCORE)CMA_END;

    /* All the records are top level (not children of other records) */
    recsIn.pRecordParent = NULL;

    /* The icons are top level */
    recsIn.zOrder = (USHORT)CMA_TOP;

    /* Redraw the container */
    recsIn.fInvalidateRecord = TRUE;

    /* Set the number of records to insert */
    recsIn.cRecordsInsert = MAXFRIENDS;

```

```

/*****
/*  Insert the records into the container.  */
*****/

WinSendMsg(hWnd,
            CM_INSERTRECORD,
            (PRECORDCORE)FirstRec,
            &recsIn);
}

=====
CONTAIN.RC
=====
#include <os2.h>
#include "contain.h"

MENU            IDM_DISPLAY
BEGIN
    MENUITEM    "Icon",        IDM_ICON
    MENUITEM    "Text",        IDM_TEXT
    MENUITEM    "Name",        IDM_NAME
END

=====
CONTAIN.H
=====
#define DLG_ADDRBOOK      100
#define CNR_ADDRBOOK      101
#define PB_ADD            102
#define PB_DIAL           103
#define PHONEBOOK         256
#define IDM_DISPLAY       400
#define IDM_ICON          401
#define IDM_NAME          402
#define IDM_TEXT          403

=====
CONTAIN.LNK
=====
contain.obj
contain.exe
contain.map
contain.def

=====
PHONES.H
=====
#define MAXFRIENDS  9

/*****
/*  This is a simple phone book database.  */
*****/
typedef struct _Phones
{
    PSZ NickName;
    PSZ FullName;
    PSZ Phone;
}PhoneBook;

/*****
/*  Normal programs would read this data from a file.  */
*****/
PhoneBook Friends[MAXFRIENDS] =
{
    "Giles",        "Kevin Giles",        "214-555-1212",
    "Bubba",        "Hank Smith",          "713-555-1212",
    "Fred",         "Fred Bicycle",        "817-555-1212",
    "Jack",         "Jack Anjill",          "919-555-1212",
    "John",         "John Richards",        "214-555-1212",
    "Toni",         "Toni Henderson",       "919-555-1212",
    "Babe",         "George Herman Ruth",    "212-555-1212",
    "Kevin",        "Kevin Kortrel",        "817-555-1212",
    "Honest Abe",   "Abraham Lincoln",     "none"
};

```

Control Windows

A *control window* is a window that an application uses in conjunction with another window to carry out simple input and output tasks. This chapter describes how to create and use control windows in PM applications.

About Control Windows

Control windows are used most often as part of a frame or dialog window, but they also can be used in a client window. An application can create control windows in a frame window by using frame-control flags in the WinCreateStdWindow function, or it can create control windows individually by calling the WinCreateWindow function.

Including control windows in a dialog window requires the use of a *dialog template*, which is a data structure that describes a dialog window and its control windows. The system uses the data in the dialog template to create the dialog window and control windows. An application can create a dialog template at run time, or it can use the system resource compiler to create a dialog-template resource.

The operating system provides many types of predefined control windows. An application can create a control of a particular type by specifying the appropriate control-window class name, either in the WinCreateWindow function or in a dialog template. The following is a list of the predefined control-window classes:

Class name	Description
WC_BUTTON	Consists of buttons and boxes the user can select by clicking the pointing device or using the keyboard.
WC_COMBOBOX	Creates a combination-box control, which combines a list-box control and an entry-field control. It allows the user to enter data by typing in the entry field or choosing from a list in the list box.
WC_CONTAINER	Creates a control for the user to group objects in a logical manner. A container can display those objects in various formats or views. The container control supports drag and drop so the user can place information in a container by simply dragging and dropping.
WC_ENTRYFIELD	Consists of a single line of text that the user can edit.
WC_FRAME	A composite window class that can contain child windows of many of the other window classes.
WC_LISTBOX	Presents a list of text items from which the user can make selections.
WC_MENU	Presents a list of items that can be displayed horizontally as action bars, or vertically as pull-down menus. Menus usually are used to provide a command interface to applications.
WC_NOTEBOOK	Creates a control for the user that is displayed as a number of pages. The top page is visible, and the others are hidden, with their presence being indicated by a visible edge on each of the back pages.
WC_SCROLLBAR	Consists of window scroll bars that let the user request to scroll the contents of an associated window.
WC_SLIDER	Creates a control that is usable for producing approximate (analog) values or

properties. Scroll bars were used for this function in the past, but the slider provides a more flexible method of achieving the same result, with less programming effort.

<code>WC_SPINBUTTON</code>	Creates a control that presents itself to the user as a scrollable ring of choices, giving the user quick access to the data. The user is presented only one item at a time, so the spin button should be used with data that is intuitively related.
<code>WC_STATIC</code>	Simple display items that do not respond to keyboard or pointing device events.
<code>WC_TITLEBAR</code>	Displays the window title or caption and lets the user move the window's owner.
<code>WC_VALUESET</code>	Creates a control similar in function to the radio buttons but provides additional flexibility to display graphical, textual, and numeric formats. The values set with this control are mutually exclusive.

A control window is always owned by another window, usually a frame or dialog window. This relationship is important because a control window sends `WM_CONTROL` messages to its owner whenever an input event occurs in the control window. Each `WM_CONTROL` message includes the identifier of the control window in which the event occurred and a notification code that specifies the nature of the event. An application specifies a control window's ID either in the `WinCreateWindow` function or in a dialog template. Each ID must be unique.

Control windows are like other predefined window classes in that they respond to standard window-management messages and functions, such as `WinSetWindowText` and `WinShowWindow`.

All control-window classes have a set of specific messages they send and receive. The summary at the end of this chapter lists the messages that all control windows have in common.

The system paints most control windows synchronously—that is, it redraws a control window as soon as any part of that window becomes invalid.

Using Control Windows

An application can use control windows in a dialog window, standard frame window, or client window. The following sections describe how to use control windows in an application.

Using Control Windows in a Dialog Window

To use a control window in a dialog window, an application specifies the control in a dialog template in the application's resource-definition file. A dialog template typically includes several control windows. When the application loads the dialog-template resource and displays the dialog window, the system automatically displays the control windows as part of the dialog window.

An application can send messages, through the dialog-window procedure, to a control window to change its state. The control window sends notification messages to the dialog-window procedure. The content of a notification message depends on the type of control window.

Using Control Windows in a Non-Dialog Window

To use a control window in a non-dialog window, an application must call the `WinCreateWindow` function, using the appropriate window

class name. An application usually specifies one of its client windows as the owner of the control window. Therefore, the client-window procedure receives notification messages from the control window. In cases where a control is owned by the frame window (such as a menu control), the notification messages to the frame window are passed to the client window.

Creating a Custom Control Window

The operating system provides the following three ways to create custom control windows:

- Use ownerdraw list boxes and menus or buttons.
- Subclass an existing control-window class.
- Register and implement a window class from scratch.

List boxes and menus can have an *ownerdraw* style, and buttons can have a user-button style, which cause the system to send a message to the owner of the ownerdraw control whenever the control must be drawn. (If the owner is a frame window, it sends these messages on to its client windows for handling by the client window procedure.) This feature lets an application alter the appearance of a control window. For menus and list boxes, the owner window draws the items within the control, and the system draws the outline of the control. For buttons, the user-button style affects the drawing of the entire control. Subclassing an existing control window is an easy way to create a custom control. The subclass procedure can alter selected behavior of the control window by processing only those messages that affect the selected behaviors. All other messages pass to the original control-window procedure.

The techniques for defining a custom control-window class are the same as those used for creating a client-window class. When you create a custom control-window class, be sure the window procedure can send and receive the messages listed in the two tables following this section.

If an application creates a private control-window class, the name of the private class could be used in the dialog template, just like a predefined window-class constant. For example, if an application defines and registers a window class called "MyControlClass", it could create a dialog window that contains that type of control window by using the following resource definition:

```
DLGTEMPLATE IDD_CUSTOM_TEST
BEGIN
  DIALOG "", IDD_CUSTOM_TEST, 1, 1, 126, 130, FS_DLGBCORDER, 0
  BEGIN
    CONTROL "This is Text", IDD_TITLE,
      37, 107, 56, 12,
      WC_STATIC,
      SS_TEXT | DT_CENTER | DT_TOP | DT_WORDBREAK
      | WS_VISIBLE
    CONTROL "Custom Control", IDD_CUSTOM,
      33, 68, 64, 13,
      "MyControlClass",
      WS_VISIBLE
    CONTROL "Okay", DID_OK,
      57, 10, 24, 14,
      WC_BUTTON,
      BS_PUSHBUTTON | BS_DEFAULT | WS_TABSTOP | WS_VISIBLE
  END
END
```

Cursors

A *cursor* is a rectangle that can be shown at any location in a window, indicating where the user's next interaction with items on the screen will happen. This chapter describes how to create and use cursors in your PM applications.

About Cursors

Only one cursor appears on the screen at a time—either marking the text-insertion point (a *text cursor*) or indicating which items the user can interact with from the keyboard (a *selection cursor*). For example, when an entry field has the keyboard focus, it displays a blinking vertical bar to show the text-insertion point; however, when a button has the keyboard focus, the cursor appears as a halftone rectangle the size of the button. The operating system draws and blinks the cursor, freeing the application from handling these details. Notice that the cursor has no direct relationship with the mouse pointer.

Cursor Creation and Destruction

The system can use only one cursor at a time, so windows must create and destroy cursors as each window gains and loses the keyboard focus. If an application attempts to use more than one cursor at a time, the results can be unpredictable and might affect other applications.

An application creates a cursor by calling `WinCreateCursor`. Generally, this is done when a window gains the keyboard focus. The application specifies the window in which to display the cursor, whether it be the desktop window, an application window, or a control window. An application destroys a cursor by calling `WinDestroyCursor`—when the specified window loses the keyboard focus, for example.

Position and Size

An application can set the position (in window coordinates) of an existing cursor by calling `WinCreateCursor`, specifying the `CURSOR_SETPOS` flag. The cursor width is usually 0 (nominal border width is used) for text-insertion cursors. This is preferable to a value of 1, since such a fine width is almost invisible on a high-resolution monitor. The cursor width also can be related to the window size—for example, when a button control uses a dotted-line cursor around the button text to indicate focus. To change the cursor size, the application must destroy the current cursor and create a new one of the desired size.

Other Cursor Characteristics

An application uses the `WinCreateCursor` function to specify information about the cursor rectangle and the clipping rectangle. `WinCreateCursor` specifies whether the cursor rectangle should be filled, framed, blinking, or halftone. In addition, the function specifies the clipping rectangle, in window coordinates, that controls the cursor clipping region. Probably the most efficient strategy is for the application to specify `NULL`, which causes the rectangle to clip the cursor to the window rectangle.

Cursor Visibility

An application can use the `WinShowCursor` function to show or hide a cursor. The operating system maintains a *show level* for the cursor: when the cursor is visible, its show level is zero; each time the cursor is hidden, its show level is incremented; each time the cursor is shown, its show level is decremented. The show:hide relationship is 1:1, so the show level cannot drop below zero. When first creating a cursor, an application should show the cursor because the application creates the cursor with a show level of 1.

The operating system automatically hides the cursor when the application calls `WinBeginPaint`; it shows the cursor when the application calls `WinEndPaint`. Therefore, there is no conflict with the cursor during `WM_PAINT` processing.

Using Cursors

This section explains how to perform the following tasks:

- Create and destroy a cursor
- Respond to a WM_SETFOCUS message

Creating and Destroying a Cursor

The following code fragment shows how an application should respond to a WM_SETFOCUS message when using a cursor in a particular window:

```
LONG    curXPos, curYPos, curWidth, curHeight;

case WM_SETFOCUS:
    if (SHORT1FROMMP(mp2)) {

        /* Gain the focus. */
        WinCreateCursor(hwnd, curXPos, curYPos, curWidth, curHeight,
            CURSOR_SOLID | CURSOR_FLASH, (PRECTL) NULL);
        WinShowCursor(hwnd, TRUE);
    }
    else {

        /* Lose the focus. */
        WinDestroyCursor(hwnd);
    }

    return 0;
```

Dialog Windows

Dialog windows (also called *dialog boxes*) provide a high-level method for applications to display and gather information. This chapter describes the creation and use of dialog windows and message boxes in your PM applications.

Note: *Dialog windows*, *dialog boxes*, and *message boxes* all are *secondary windows* to the user.

About Dialog Windows

A dialog window is a temporary window that contains one or more control windows and, typically, is used to display messages to and gather input from the user. An application usually destroys a dialog window immediately after using it.

The OS/2 operating system contains many functions and messages that help manage the control windows that make up a dialog window, thereby easing the burden of maintaining complex input and output systems.

Modal and Modeless Dialog Windows

Dialog windows can be modal or modeless. A *modal* dialog window requires that the dialog window be dismissed before the user can activate other windows in the same application. Generally, an application uses a modal dialog window to get essential information from the

user before proceeding with an operation. A *modeless* dialog window allows the user to activate other windows in the same application without dismissing the dialog window. Both modal and modeless dialog windows allow the user to activate windows in another application before responding to the dialog window.

Modal dialog windows are easier for an application to manage because they are created, perform their task, and are closed, all with a single function call.

Modeless dialog windows require more attention from the application because they exist until explicitly dismissed. Modeless dialog windows provide a more flexible interface, however, by allowing the user to move to other windows in the application before responding to the dialog window.

Dialog Items

A *dialog item* is a child window of the dialog window, which usually is a window of class `WC_FRAME`. The operating system provides many predefined window classes, called *control windows*, that you can use as dialog items. Predefined control windows include static display boxes, text-entry fields, buttons, and list boxes. You also can use customized window classes as dialog items.

Dialog items are windows and, thus, can be manipulated by all window-management functions relating to size, position, and visibility. Dialog items always are owned by the dialog frame window. Most predefined control-window classes send notification messages to their owners when the user interacts with their control windows. The dialog frame window receives these notification messages and passes them to the application through the application-defined dialog procedure.

Dialog-Item Groups

Items within a dialog window can be organized into *dialog-item groups*. When items are arranged in a group, the user can move from one item to another in the same group by using the direction keys. When the user presses a direction key, the focus will not shift to items in other groups within the dialog window.

Arranging items in groups is useful for radio buttons and check boxes. Although some control types also can be displayed this way, entry-field controls cannot; they process direction keys themselves, as do MLE, value-set, container, slider, and notebook controls.

The first item in a dialog-item group has the `WS_GROUP` window style. All subsequent items in the dialog template are considered part of that group until another item is given the `WS_GROUP` style, which begins a new group.

The `WS_TABSTOP` style often is used along with the `WS_GROUP` style. `WS_TABSTOP` marks the items that can receive the focus when the user presses the Tab key. Each time the user presses the Tab key, the focus moves to the next item that has the `WS_TABSTOP` style. Generally, the `WS_GROUP` and `WS_TABSTOP` styles are defined together for the first item of each group in the dialog template. This makes it possible for a user to press the Tab key to move among groups of items and to use the direction keys to move among items in a group.

The `WS_TABSTOP` style should not be used for radio buttons because the system automatically maintains a tab stop on any selected item in a radio-button group; therefore, when the Tab key is pressed in a group of radio buttons, the focus remains on the currently selected item.

The `WS_GROUP` and `WS_TABSTOP` styles are also useful for preventing the user from moving to a particular button when using the keyboard. For example, if the dialog window has **OK** and **Cancel** push buttons, they should be in the same group, with the **OK** push button as the first item in the group. The user can press Tab to select the **OK** push button but not the **Cancel** push button. To move to the **Cancel** button using the keyboard, the user first must press the Tab key to move to the **OK** push button, and then press a direction key to move the focus to the **Cancel** push button.

Message Boxes

Message boxes are dialog windows predefined by the system and used as a simple interface for applications, without the necessity of creating dialog-template resources or dialog procedures. Message boxes are best for short notification messages that require a simple acknowledgment or choice by the user. Applications do not specify a dialog procedure for message boxes, so they cannot readily change the action of a message box. However, there is no need to do so, since there are many predefined message-box styles. There are two types of message boxes available to applications: standard and enhanced.

Standard Message Boxes

To generate a standard message box, an application calls `WinMessageBox` and specifies the type of message box and message text. The system displays the message and waits for the user to dismiss the message box by selecting a button in the message box. The system then returns a result code to the application, indicating which button the user selected.

Standard message boxes are always modal—either application-modal or system-modal. *Application-modal* (the default style) means that the user cannot activate another window in the current application before responding to the message box but can switch to another application. *System-modal* means that the user cannot activate another window in any application before responding to the message box. A system-modal message box should be used only to display urgent error messages (running out of memory, for example).

Enhanced Message Boxes

To generate an enhanced message box, an application calls `WinMessageBox2`. An enhanced message box has all the functionality of the standard message box, with the addition of the following:

- It can be modeless.
- Its buttons can be customized with text and icons.
- It can support customized icons in the window icon field.

In creating enhanced message boxes, the `MB2INFO` button information block structure is used to specify button style flags as shown in the following table:

Style Name	Description
<code>MB_APPLMODAL</code>	Message box is application modal. This is the default case.
<code>MB_CUSTOMICON</code>	A user-specified value.
<code>MB_ERROR</code>	Message box contains a small red circle with a red line across it.
<code>MB_ICONASTERISK</code>	Message box contains an information (i) icon.
<code>MB_ICONEXCLAMATION</code>	Message box contains an exclamation point (!) icon.
<code>MB_ICONHAND</code>	Message box contains a small red circle with a red line across it.
<code>MB_ICONQUESTION</code>	Message box contains a question mark (?) icon.
<code>MB_INFORMATION</code>	Message box contains an information (i) icon.
<code>MB_MOVEABLE</code>	Message box is moveable. A title bar and system menu with Move, Close and Task Manager choices are displayed.
<code>MB_NOICON</code>	Message box is not to contain an icon.
<code>MB_NONMODAL</code>	Message box is modeless (the program continues after displaying the message box).
<code>MB_QUERY</code>	Message box contains a question mark (?) icon.

<code>MB_SYSTEMMODAL</code>	Message box is system modal.
<code>MB_WARNING</code>	Message box contains an exclamation point (!) icon.

Minimizing Dialog Windows

Whenever the dialog window is to be minimized to the desktop (as opposed to the Minimized Window Viewer), the program should hide all its child control windows, then restore them when the dialog needs to be restored.

Dialog Data Structures

Each item in a dialog window is described by a `DLGITEM` data structure. This structure is rarely accessed directly by an application, since system functions handle most of the manipulation of dialog items. Applications that create dialog items that are not defined as part of a dialog-template resource must create dialog-window-item structures in memory.

A dialog window can have many items, so applications can use another structure, `DLGTEMPLATE`, to define the items. This structure consists of header information, followed by an array of dialog-window items. Applications that create dialog windows without using dialog resources must create a dialog template in memory, and, then, call the `WinCreateDlg` function.

Dialog Resources

Most applications define dialog templates in resource files rather than constructing template data structures in memory at run time. The dialog resource file defines the size and style of the dialog-window frame and specifies each dialog item.

The dimensions and position of each dialog item are specified in dialog coordinates, which are based on the size of the system font. A horizontal unit is one-fourth the average width of the characters in the system font; a vertical unit is one-eighth the average height of the characters in the system font. The origin of the dialog template is the lower-left corner of the dialog window. The operating system provides the `WinMapDlgPoints` function for converting dialog coordinates into window coordinates.

Using Message Boxes and Dialog Windows

The simplest dialog window is the message box. Most message boxes present simple messages and offer the user one, two, or three responses (represented by buttons). A message box is easy to use and is appropriate when an application requires a clearly defined response to a static message. However, standard message boxes lack flexibility in size and placement on the screen and are limited in the choices they offer the user. Applications that require more control over the size, position, and content should use enhanced message boxes or regular Dialog Windows instead of standard message boxes.

Creating a Standard Message Box

There are three parts to a message box: the icon, the message, and buttons. Applications specify the icons and buttons by using

message-box style constants. Message text is specified by a null-terminated string.

To create a message box, the application calls `WinMessageBox`, which displays the message box and processes user input until the user selects a button in the message box. The `WinMessageBox` return value indicates which button the user selected.

The following code fragment illustrates how to create a message box with a default **Yes** button, a **No** button, and a question-mark (?) icon. This example assumes that you have defined a string resource with the `MY_MESSAGESTR_ID` identifier in the resource file.

```
UCHAR  szMessageString[255];
ULONG  ulResult;

WinLoadString(hab, (HMODULE) NULL, MY_MESSAGESTR_ID,
              sizeof(szMessageString), szMessageString);

ulResult = WinMessageBox(hwndFrame, /* Parent      */
                          hwndFrame, /* Owner      */
                          szMessageString, /* Text      */
                          (PSZ) NULL, /* caption   */
                          MY_MESSAGEWIN, /* Window ID */
                          MB_YESNO | /* Style     */
                          MB_ICONQUESTION | /* Style     */
                          MB_DEFBUTTON1); /* Style     */

if (ulResult == MBID_YES) {
    /* Do yes case. */
} else {
    /* Do no case. */
}
```

The `WinMessageBox` function returns predefined values indicating which button has been selected.

Notice that strings for message boxes should be defined as string resources to facilitate program translation for other countries. However, there is danger in using string resources in message boxes that are called in low-memory situations; loading a string resource in such situations could result in severe memory problems and cause an application to fail. One way to prevent this problem is to preload the string resource and make it nondiscardable so it will be available when the message box must be displayed.

Creating a System-Modal Standard Message Box

There are two levels of modality for system-modal message boxes—*soft* modal and *hard* modal. A soft-modal message box does not allow keystrokes or mouse input to reach any other window but does allow other messages, such as deactivation and timer messages, to reach other windows. A hard-modal message box does not allow any messages to reach other windows. A hard-modal message box is appropriate for serious system warnings.

To create a hard-modal message box, combine the `MB_ICONHAND` style with the `MB_SYSTEMMODAL` style. To create a soft-modal message box, use the `MB_SYSTEMMODAL` style with any style other than `MB_ICONHAND`. The `MB_SYSTEMMODAL` icon always is in memory and is available even in low-memory situations.

Creating an Enhanced Message Box

`WinMessageBox2` creates a message window that can be used to display error messages and ask questions. It is a more powerful version of `WinMessageBox`, including options for non-modality and customization of buttons with text and icons or mini-icons. Buttons included in the enhanced message box are specified in the button definition array `MB2D`, where custom text can be added.

To support the use of the `MB_NONMODAL` style, two notification messages are used:

WM_MSGBOXINIT	This message notifies the owner of the message when a non-modal message box is being displayed. It is the responsibility of the owner window to store the window handle returned by the function for later use when the message box is to be destroyed.
WM_MSGBOXDISMISS	This message notifies the owner of the message when a non-modal message box has been dismissed. It is the parent window's responsibility to destroy the message box.

The following example uses WinMessageBox2 to create a message box containing a customized icon:

```
#define INCL_WINDIALOGS          /* Window Dialog Manager Functions */
#define INCL_WINPOINTERS        /* Window Pointer Functions */

#include <os2.h>
#include <stdio.h>
#include <string.h>

CHAR      szMsg[100];           /* Message */
HWND      hwndClient;           /* Client-window handle */
MB2INFO   mb2info;              /* Message Box input structure */

MB2D mb2d[4] = {                /* Array of button definitions*/
    { "AAAA", ID_BUTTON1, BS_DEFAULT},
    { "BBBB", ID_BUTTON2, 0},
    { "CCCC", ID_BUTTON3, 0},
    { "DDDD", ID_BUTTON4, 0}
};

mb2info.hIcon = WinLoadPointer(HWND_DESKTOP, 0, ID_ICON1);
mb2info.cButtons = 4;           /* Number of buttons */
mb2info.flStyle = MB_CUSTOMICON | MB_MOVEABLE;
/* Icon style flags */
mb2info.hwndNotify = NULLHANDLE; /* Reserved */
mb2info.cb = sizeof(MB2INFO) + ((mb2info.cButtons > 1) ?
    (mb2info.cButtons - 1) * sizeof (MB2D) : 0);

memcpy (&mb2info.mb2d, &mb2d, mb2info.cb);

mb2info.pmb2d = mb2d;          /* Array of button definitions*/

sprintf (&szMsg, %s, "Error condition exists");

WinMessageBox2(HWND_DESKTOP,
    hwndClient,                /* Client-window handle */
    &szMsg,                     /* Body of the message */
    "Debugging Information",
    /* Title of the message */
    0,                         /* Message Box id */
    &mb2info);                 /* Message Box input structure */
```

Using a Dialog Window

When using a dialog window, an application must load the dialog window, process user input, and destroy the dialog window when the user finishes the task. The process for handling a dialog window varies, depending on whether the dialog window is modal or modeless.

Creating a Dialog Template

The following source-code fragment creates a dialog template. Notice that the WS_GROUP and WS_TABSTOP style designations are

given for the first item in each group.

```
DLGTEMPLATE IDD_ABOUT
BEGIN
    DIALOG "", IDD_ABOUT2,
        10, 10, 150, 110, FS_DLGBOARDER, 0
    BEGIN
        CONTROL "Attributes:", 100,
            10, 30, 100, 70,
            WC_STATIC,
            SS_GROUPBOX | WS_VISIBLE
        CONTROL "Highlighted", 101,
            20, 80, 58, 12,
            WC_BUTTON,
            WS_GROUP | WS_TABSTOP | BS_AUTOCHECKBOX | WS_VISIBLE
        CONTROL "Enabled", 102,
            20, 60, 58, 12,
            WC_BUTTON,
            BS_AUTOCHECKBOX | WS_VISIBLE
        CONTROL "Checked", 103,
            20, 40, 58, 12,
            WC_BUTTON,
            BS_AUTOCHECKBOX | WS_VISIBLE
        CONTROL "Okay", DID_OK,
            10, 10, 50, 14,
            WC_BUTTON,
            WS_GROUP | WS_TABSTOP | BS_PUSHBUTTON | BS_DEFAULT | WS_VISIBLE
        CONTROL "Cancel", DID_CANCEL,
            80, 10, 50, 14,
            WC_BUTTON,
            BS_PUSHBUTTON | WS_VISIBLE
    END
END
```

Creating a Modal Dialog Window

The easiest way to use a modal dialog window is to define a dialog template in the resource file (as in the preceding section), and then, call the `WinDlgBox` function, specifying the dialog-window resource identifier and a pointer to the dialog procedure. `WinDlgBox` loads the dialog-window resource, displays the dialog window, and handles all user input until the user dismisses the dialog window. The dialog procedure receives messages when the dialog window is created (`WM_INITDLG`) and other messages each time the user interacts with a dialog item (enters text in entry fields or selects a button, for example).

You must specify both the parent and owner windows when loading a dialog window using the `WinDlgBox` function. Generally, the parent window will be `HWND_DESKTOP` and the owner will be a client window in your application.

Dialog windows typically contain buttons that send `WM_COMMAND` messages when selected by the user. `WM_COMMAND` messages passed to the `WinDefDlgProc` function result in the `WinDismissDlg` function's being called, with the window identifier of the source button as the return code (from `WinDismissDlg`). Dialog windows with either **OK** or **Cancel** as their only button can ignore `WM_COMMAND` messages, allowing them to be passed to `WinDefDlgProc`. `WinDefDlgProc` calls `WinDismissDlg` to dismiss the dialog window and returns the `DID_OK` or `DID_CANCEL` code.

Passing `WM_COMMAND` messages to `WinDefDlgProc` means that all button presses in the dialog window dismiss the dialog window. If you want certain buttons to initiate operations without closing the dialog window, or if you want to perform some processing without closing the dialog window, handle the `WM_COMMAND` messages in the dialog procedure.

If you handle `WM_COMMAND` messages in the dialog procedure, you must call `WinDismissDlg` to dismiss the dialog window. Your dialog procedure passes the `DID_OK` code to `WinDismissDlg` if the user selects the **OK** button or the `DID_CANCEL` code if the user selects the **Cancel** button.

When you call `WinDismissDlg` or pass the `WM_COMMAND` message to `WinDefDlgProc`, the dialog window is dismissed, and the `WinDlgBox` function returns the value passed to `WinDismissDlg`. This return value identifies the button selected.

An alternative to using `WinDlgBox` is to call the individual functions that duplicate its functionality, as shown in the following code fragment:

```

HWND  hwndDlg;
ULONG ulResult;

hwndDlg = WinLoadDlg(...);
ulResult = WinProcessDlg(hwndDlg);
WinDestroyWindow(hwndDlg);

```

After calling the `WinProcessDlg` function, your dialog procedure must call `WinDismissDlg` to dismiss the dialog window. Although the dialog window is *dismissed* (hidden), it still exists. You must call the `WinDestroyWindow` function to destroy a dialog window if it was loaded using the `WinLoadDlg` function. `WinDlgBox` automatically destroys a dialog window before returning.

If you want to manipulate individual items in a dialog window, or add a menu after loading the dialog window (but before calling `WinProcessDlg`), it is better to make individual calls rather than call `WinDlgBox`. Individual calls also are useful for querying individual dialog items—to determine the contents of an entry-field control after a dialog window is closed but before it is destroyed, for example. Destroying a dialog window also destroys any dialog-item control windows that are child windows of the dialog window.

Creating a Modeless Dialog Window

To use a modeless dialog window in an application, create a dialog template in the resource file, just as for a modal dialog window. Modeless dialog windows share the screen equally with other frame windows. It is a good idea to give modeless dialog windows a title bar so they can be moved around the screen. The following Resource Compiler source-code fragment shows a dialog template for a dialog window with a title bar, system menu, and minimize button.

```

DLGTEMPLATE IDD_SAMP
BEGIN
    DIALOG "Modeless Dialog", IDD_SAMP, 80, 92, 126, 130,
        WS_VISIBLE | FS_DLGBOARDER,
        FCF_TITLEBAR | FCF_SYSMENU | FCF_MINBUTTON

    BEGIN

        /* Put control-window definitions here. */

    END
END

```

The application loads the dialog resource from the resource file using the `WinLoadDlg` function, receiving in return a window handle to the dialog window. The application treats the dialog window as if it were an ordinary window. Messages for the dialog window are dispatched through the event loop the application uses for its other windows. In fact, an application can have a modeless dialog window as its only window.

The resource for a modeless dialog window is like the resource used for a modal dialog window. The difference between modal and modeless dialog windows is the way applications handle input to each. For a modal dialog, the `WinDlgBox` and `WinProcessDlg` functions handle all user input to the dialog window, preventing access to other windows in the application. For a modeless dialog window, the application does not call these functions, relying instead on a normal message loop to dispatch messages to the dialog procedure.

The primary difference between a modeless dialog window and a standard frame window with child control windows is that, for a modeless dialog window, an application can define child windows for the dialog window in a dialog template, automating the process of creating the window and its child windows. The same effect can be achieved by creating a standard frame window, but then, the child control windows must be created individually.

It is important that an application keep track of all open modeless dialog windows so that it can destroy all open windows before terminating.

Initializing a Dialog Window

Generally, an application defines a dialog template in its resource file and loads the dialog window by calling the `WinLoadDlg` function or the `WinDlgBox` function (which calls `WinLoadDlg`). The dialog window is created as an invisible window unless the window style `WS_VISIBLE` is specified in the dialog template. A `WM_INITDLG` message is sent to the dialog procedure before `WinLoadDlg` returns. As each control defined in the template is created, the dialog procedure might receive various control notifications before the function returns. `WinLoadDlg` returns a handle to the dialog window immediately after creating a dialog window.

In general, it is a good idea to define a dialog window as invisible, since this allows for optimization. For example, an experienced user might type ahead rapidly, anticipating the processing of a dialog-window command. In such a case, there is no need to display the dialog window, because the user has finished the interaction before the window can be displayed. This is how the `WinProcessDlg` function works—it does not display a dialog window while there still are `WM_CHAR` messages in the input queue; it processes the `WM_CHAR` messages before displaying the dialog window.

As control windows in a dialog window are created from the template, strings in the template are processed by the `WinSubstituteStrings` function. Any `WM_SUBSTITUTESTRING` messages are sent to the dialog procedure before `WinLoadDlg` returns.

When child windows of a dialog window are created, `WinSubstituteStrings` is used so child windows can make substitutions in their window text. If any child-window text string contains the percent sign (%) substitution character, the length of the text string is limited to 256 characters after it is returned from the substitution.

Adding a Menu in a Dialog Window

To create a menu bar and menus in a dialog window, an application first must load the dialog window to get a handle to the dialog-frame window. The dialog-frame window can be associated with a menu resource by calling the `WinLoadMenu` function. This function requires arguments that specify the menu identifier and the handle of the parent window for the menu. Finally, the dialog-frame window must incorporate the menu by sending a `WM_UPDATEFRAME` message to the dialog window. The following code fragment illustrates these operations:

```
HWND hwndDialog, hwndMenu;

/* Get the dialog resource. */
hwndDialog = WinLoadDlg(...);

/* Get the menu resource and attach it to the dialog window. */
hwndMenu = WinLoadMenu(hwndDialog, ...);

/* Inform the dialog window that it has a new menu. */
WinSendMsg(hwndDialog, WM_UPDATEFRAME, (MPARAM) NULL, (MPARAM) NULL);
```

Applications can create menus in both modal and modeless dialog windows. The preceding code fragment can be used for either type of dialog window. For a modal dialog window, your application must call the `WinProcessDlg` function to handle user input until the dialog window is dismissed. For a modeless dialog window, your application must call the `WinShowWindow` function to display the dialog window, enabling the message loop to direct messages to the dialog window.

Creating a Dialog Procedure

In contrast to window procedures, which receive `WM_CREATE` messages, dialog procedures receive `WM_INITDLG` messages, which are sent after a dialog window is created, but before it is displayed. `WM_INITDLG` can do the same type of initialization tasks that `WM_CREATE` handles, but is not the first message that is received.

For example, if a dialog window contains a list box, use WM_INITDLG to fill the list box with items. Also use this procedure to enable or disable buttons in a dialog window, depending on your application.

You also can call the WinSetDlgItemText or WinSetDlgItemShort functions during dialog initialization, to set up text items that reflect the current conditions in your application.

Another typical task for the WM_INITDLG message handler is centering a dialog window on the screen or within its owner window. The following code fragment illustrates how to center a dialog window on the screen using WM_INITDLG:

```
RECTL rclScreen,rclDialog;
LONG  sWidth,sHeight,sBLCx,sBLCy;

case WM_INITDLG:
    /* Center the dialog window and get the screen rectangle. */
    WinQueryWindowRect(HWND_DESKTOP, &rclScreen);

    /* Get the dialog-window rectangle. */
    WinQueryWindowRect(hwnd, &rclDialog);

    /* Get the dialog-window width. */
    sWidth = (LONG) (rclDialog.xRight - rclDialog.xLeft);

    /* Get the dialog-window height. */
    sHeight = (LONG) (rclDialog.yTop - rclDialog.yBottom);

    /* Set the horizontal coordinate of the lower-left corner. */
    sBLCx = ((LONG) rclScreen.xRight - sWidth) / 2;

    /* Set vertical coordinate of the lower-left corner. */
    sBLCy = ((LONG) rclScreen.yTop - sHeight) / 2;

    /* Move, size, and show the window. */
    WinSetWindowPos(hwnd,
        HWND_TOP,
        sBLCx, sBLCy,
        0, 0, /* Ignores size arguments */
        SWP_MOVE);

    return 0;
```

The dialog procedure receives notification messages from each control-window item in a dialog window whenever a user clicks an item or enters text in an entry field. Most dialog procedures wait for the user to select one or more dialog-window buttons to signal being finished with the dialog window. When the dialog procedure receives one of these messages, it calls the WinDismissDlg function, as shown in the following code fragment. The second argument to WinDismissDlg is the value returned by the WinDlgBox or WinProcessDlg functions. Generally, these functions return the identifier of the button that was pressed.

```
MRESULT EXPENTRY SampDialogProc(HWND hwnd,
                                ULONG ulMessage,
                                MPARAM mp1,
                                MPARAM mp2)
{
    switch (ulMessage) {
        case WM_COMMAND:
            switch (SHORT1FROMMP(mp1)) {
                case DID_OK:
                    /*
                     * Final dialog-item queries,
                     * dismiss the dialog.
                     */
                    WinDismissDlg(hwnd, DID_OK);
                    return 0;
            }
            break;
    }
    return (WinDefDlgProc(hwnd, ulMessage, mp1, mp2));
}
```

Other dialog-window items send notification messages specific to the type of control window. Your dialog procedure should respond to notification messages from any relevant or important dialog items, and pass the messages that your dialog procedure does not handle to the WinDefDlgProc function for default processing. The default dialog procedure is used similarly to the default frame-window procedure.

The WM_COMMAND message from the **OK** button indicates that the user has selected the **OK** button and is finished with the dialog window. If the dialog window has other controls, such as entry fields or check boxes, have your dialog procedure query the contents or state of each control upon receipt of a message from the **OK** button. Before dismissing a dialog window, have your dialog procedure collect input from each dialog-window control before closing the dialog window.

Manipulating Dialog Items

Dialog items are control windows and, as such, can be manipulated using standard window-management function calls. The window handle is obtained for each dialog item by calling the WinWindowFromID function and passing the window handle for the dialog window and the window identifier for the dialog item as defined in the dialog template. Include the following Resource Compiler source-code fragment in your dialog template:

```
DLGTEMPLATE IDD_ABOUT
BEGIN
    DIALOG "", IDD_ABOUT, 80, 92, 126, 130, FS_DLGBOARDER, 0
    BEGIN
        PUSHBUTTON "My Button", ITEMID_MYBUTTON, 37, 107, 56, 12

        /* Other item definitions ... */
    END
END
```

Based on this code fragment, your application will receive the button-item handle by initiating the following call to WinWindowFromID:

```
hwndItem = WinWindowFromID(hwndDialog, ITEMID_MYBUTTON);
```

Applications often change the contents, enabled state, or position of dialog items at run time. For example, in a dialog window that contains a list box of file names and an **Open** button, the **Open** button should be disabled until the user selects a file from the list. To do this, define the button as disabled in the dialog resource so that it is disabled when the dialog window first is displayed. At run time, the dialog procedure receives a notification message from the list box when the user selects a file. At that time, the dialog procedure should call the WinEnableWindow function to enable the **Open** button.

Applications also can change the text in static dialog items and buttons by calling the WinSetWindowText function and using the window handle of a particular dialog item.

Direct Manipulation

Direct manipulation is the act of moving graphical representations such as OS/2 icons around the screen using a pointing device, such as a mouse. This chapter explains how to use direct manipulation in PM applications.

About Direct Manipulation

The direct manipulation protocol enables the user to select an object in a window, drag it to another location, and drop it on another object or in another window. *Dragging* is the act of moving an object as though it were attached to the pointer; it is performed by pressing and holding the drag button and moving the pointer. *Dropping* is the act of fixing the position of the dragged object by releasing the drag button on the pointer. This causes interaction (data exchange) between the window from which the selected object is dragged and the window containing the object on which the selected object is dropped.

The window containing the dragged object is the *source*. The window containing the object that was dropped on is the *target*. The source and target can be the same window, different windows within the same application, or windows belonging to different applications. The dragged object can be either the only visible object in the source window or one of many objects. The target object can be either the only visible object in the target window or one of many objects. A source or target window that contains multiple objects is a *container window*.

The data exchange that occurs between the source and target after a direct manipulation operation enables applications that support the protocol to integrate easily, while providing a simple user interface.

Application-Defined Drag Operations

At times it may be useful for an application to define its own drag operation to facilitate functions between two windows in the same application or between closely related applications. For example, an application implementing a keyboard remapping function may want to provide a method of redefining keys with direct manipulation. This application could define an operation whereby dragging one key to another exchanges the definitions of the two keys. The protocol provides the extendability to enable this kind of function.

Rendering Mechanism and Format

The *rendering mechanism* represents the way in which you want to exchange the data, for example, dynamic data exchange (DDE). The *rendering format* identifies the actual type or true type of the data, for example, text. To exchange data, both the source and target must know how to communicate with each other through the rendering mechanism and understand the particular format of the data.

The *native rendering mechanism* and format of the object is the mechanism that most naturally conveys the data, either where it is now, or where it can be put most easily. The format conveys all information about the data. For example, a spreadsheet cell has a location in a row and column of a spreadsheet. Rendering the spreadsheet cell in a simple text format would cause this information to be lost, so a more appropriate format should be chosen for its native rendering format.

A source application may be able to exchange data with a target through several mechanisms, such as:

- Dynamic Data Exchange (DDE)
- OS/2 File
- Print

Additionally, the source application might be able to *render* the data in various formats, that is, into various types. For example, a spreadsheet application could render its contents in a spreadsheet format or into a simple text format. The ability of the source application to render the data in some format might, itself, depend on the exchange mechanism used. The rendering mechanisms and formats that a source application can support, for each object dropped, are provided to the target through the *hstrRMF* field in the DRAGITEM data structure.

The target application may also be able to exchange data with the source through several different combinations of mechanism and format. The target is responsible for obtaining the data from the source in the format that they both support and that provides the highest level of information about the data.

While making this determination, the target must consider the exchange capabilities offered by the mechanism. For example, an OS/2 File exchange mechanism can provide only a snapshot of the data at the time the direct manipulation operation occurred. An exchange using DDE, on the other hand, offers the target an opportunity to remain informed about changes to the data.

Non-Standard Rendering Mechanisms

Some standard rendering mechanisms are already defined, but the system lets the set of rendering mechanisms be expanded, allowing for:

- Additional standard rendering mechanisms to be defined in the future
- Application definition of private or nonstandard rendering mechanisms

An application can elect to support some, all, or none of the standard rendering mechanisms defined by the system. Applications that do not support any of the standard rendering mechanisms are not precluded from using direct manipulation. However, support of the standard rendering mechanisms and formats increases the chances of a successful data transfer between applications.

An application that supports a particular rendering mechanism, whether or not it is a rendering mechanism defined by the system, must follow a specific set of guidelines defined by that rendering mechanism, including conversation-initiation procedures and naming conventions.

Responsibilities of a Source Application

The source is responsible for starting a direct manipulation operation. Startup can be accomplished only with a pointing device, such as a mouse. The operation starts when the application detects that a drag button has been pressed and the pointing device has moved. Dragging continues until terminated, which is usually when the button is released.

Although the direct manipulation protocol lets the application use any button for dragging, it is recommended that the system-defined drag button be used for direct manipulation operations.

The source has the following responsibilities in preparing for the actual drag of the objects across the screen:

- Allocate and initialize the DRAGINFO data structure that conveys the necessary information about each object to the target.
- Initialize a set of DRAGIMAGE data structures that describe the image to be displayed during the drag operation.
- Make the following information known to the system:
 - The type of each object being directly manipulated
 - The rendering mechanism and format for each object
 - The suggested name of the object at the target
 - The name of the container or folder containing the source object
 - The name of the object at the source
 - The *true* type of each object being directly manipulated
 - The native rendering mechanism and format for each object

Responsibilities of a Target Application

The target in a direct manipulation operation is responsible for determining whether a particular set of objects can be dropped on it, and for providing the user with visible cues regarding the operation. A target is informed of the operation through messages sent to it as the pointer, provided by the source, is dragged across the screen.

When a set of objects is dropped on the target, the target is responsible for establishing the appropriate conversations with the source to accomplish the data transfer. The type of conversation for each object is based on the rendering mechanism and format of the object being dropped.

The target application is responsible for:

- Determining if data can be exchanged between source and target by verifying that both applications share knowledge of at least one rendering mechanism and format

- Providing visible feedback, or target emphasis, on whether a drop is allowed
- Defining the default state of a direct manipulation operation
- Initiating conversations with the source for data transfer

Messages Sent to a Target Application

The following table describes the messages that are sent to each window whose boundaries are crossed as the user drags the object around the screen:

Message Name	Description
DM_DRAGOVER	Sent to the window under the pointer as the pointer is dragged across it. A single DM_DRAGOVER message is sent each time the pointer moves and each time a key is pressed or released, and it contains a pointer to the DRAGINFO data structure. The target can access this data structure with DrgAccessDraginfo.
DM_DRAGLEAVE	Sent whenever the DM_DRAGOVER message has been sent to a window, and the pointer is moved outside the bounds of that window. If the target or an object in the window had been emphasized as a target, it should be de-emphasized.
DM_DROP	Sent to the target to provide it with the information necessary to establish a conversation for data exchange with the source. The target should immediately remove any target emphasis. The data transfers must not be done before responding to the DM_DROP message.
DM_DROPHELP	Posted to a target to indicate that the user requested help for the drag operation while over that target.

Response to Messages Sent to a Target Application

The following table shows the four possible responses available to the target when it receives a DM_DRAGOVER message. The target sends these values to the window handle specified in the DRAGINFO data structure.

Message Name	Description
DOR_DROP	Sent if the objects being dragged are acceptable. A drop does not occur unless DOR_DROP is returned.
DOR_NODROP	Sent if the objects being dragged are acceptable and the target supports the current operation, but the objects cannot be dropped on the current location in the target window. For example, a list box might return DOR_NODROP if it contains objects that can be dropped on, but the pointer is

over an object that cannot be dropped on.
 If the target response is DOR_NODROP, the DM_DRAGOVER message continues to be sent to the target when:

- o The pointer is moved
- o A keyboard key is pressed
- o The pointer is moved out of and back into the window.

DOR_NODROPOP Sent if the objects being dragged are acceptable, but the target does not support the current operation. This response implies that the drop may be valid if the drag operation changes. For example, copying a file to a shredder would not be valid, but moving a file to a shredder would be. Once the target has sent DOR_NODROPOP, no further DM_DRAGOVER messages is sent to the target until:

- o A keyboard key is pressed
- o The pointer is moved out of and back into the window.

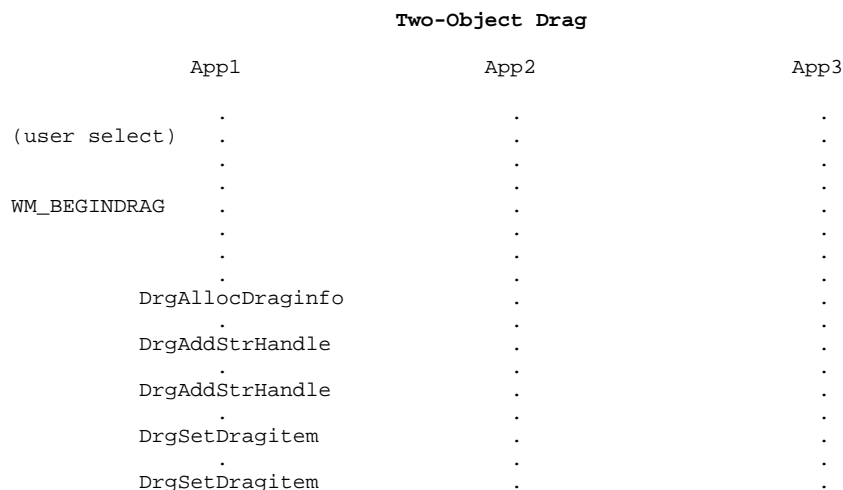
DOR_NEVERDROP Sent when the objects being dragged are not acceptable, and the target will never accept them. Once the target has sent DOR_NEVERDROP, no further DM_DRAGOVER messages are sent to that target until the pointer is moved out of and back into the target window.

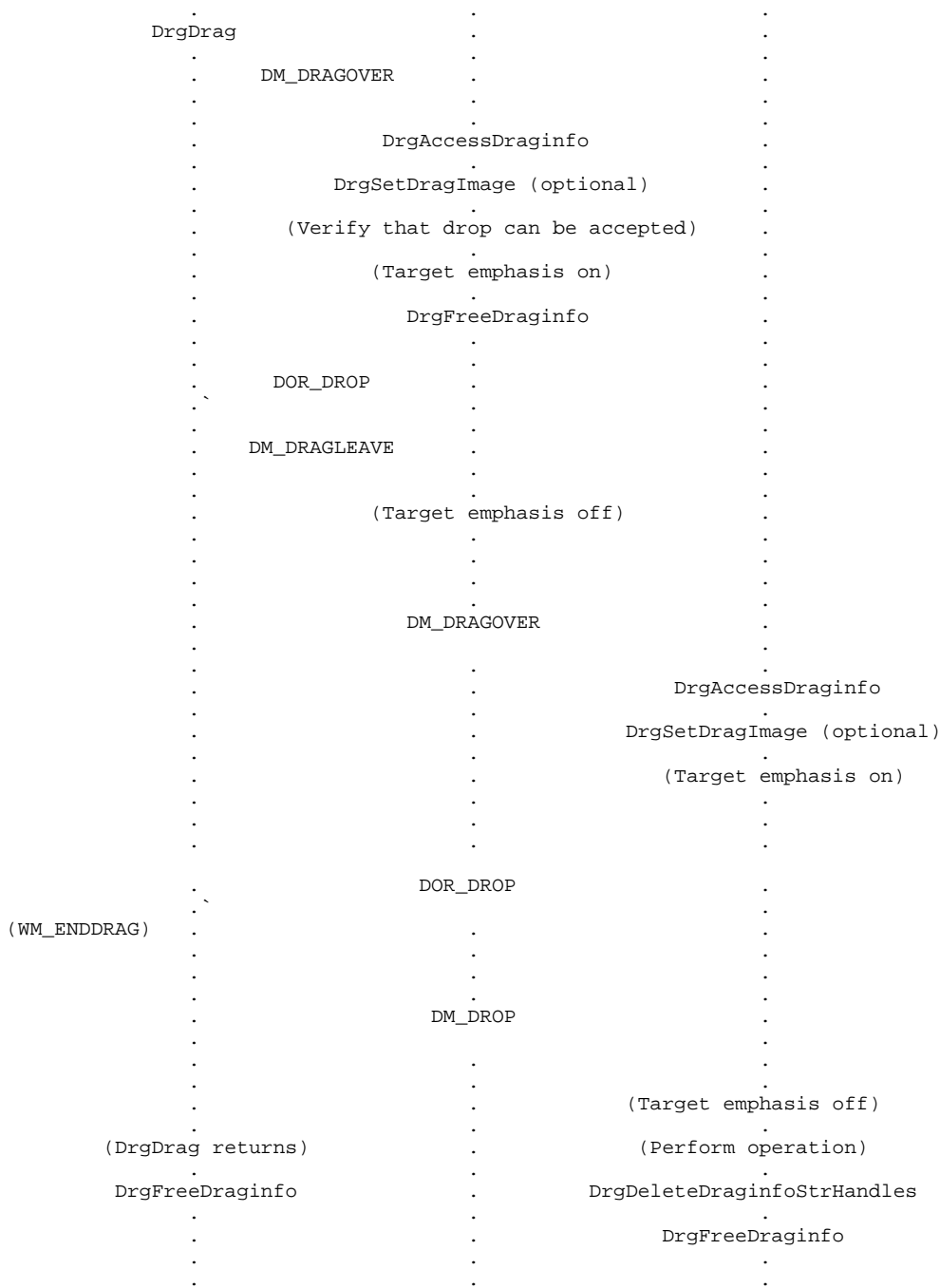
If a reply other than DOR_DROP is received from a target, the augmentation emphasis is automatically changed to indicate that no drop is allowed. This gives the user a visible cue that a drop cannot occur. The emphasis is reverted to *drop allowed* when a DOR_DROP reply is received from some target.

Two-Object Drag Operation

The following diagram represents the sequence of functions and message flows for a typical direct manipulation operation. The flow shows a two-object drag from App1 to App3, dragging over App2.

The direct manipulation operation is started by the source window procedure after the user selects the objects to be manipulated and the source receives a WM_BEGINDRAG message.

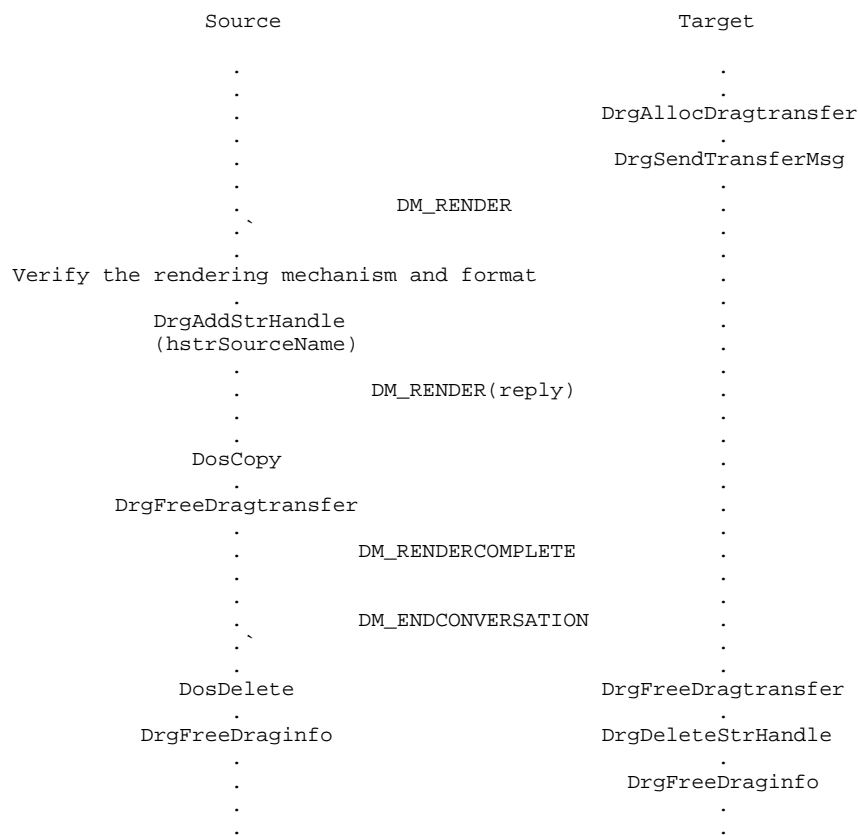




Conversation after the Drop

The following diagram represents the sequence of message flows for a typical direct manipulation data-transfer operation. The flow describes a single-object move from source to target. The user dropped on white space in the target container.

For this example, assume that the rendering mechanism selected is DRM_OS2FILE and that the source does not initially provide the target with the source item's file name. Also assume that the source and target items exist on different drives.



Canceling a Drag Operation

The user can end a direct manipulation operation in one of the three following ways:

- Pressing the Esc key to cancel the operation
- Releasing the drag button when the pointer is over a target that cannot accept the drop

This action is equivalent to pressing the Esc key. When the pointer is over a target that can accept the drop, the target is informed of the drop, and the source is given the window handle of the target.

- Pressing the F1 key to request help

A DM_DROPHELP message is posted to the target. This enables the target to provide the user with assistance regarding:

- What would happen if the user dropped the object on that target
- Why the target cannot accept a particular drop

The source sees this termination of the direct manipulation operation as a cancelation.

About Pickup and Drop

Pickup and Drop (also known as Lazy Drag) enables a drag operation to occur without requiring that the drag button be pressed for the duration of the operation (as in the standard direct manipulation operation). Pickup and Drop is non-modal in nature, allowing the user to interrupt the drag operation with other processes, and eliminating the requirement that both the source and target objects be visible prior to initiation of the drag operation (as in standard protocol). Pickup and Drop does not replace the standard, modal direct manipulation operation; it offers a more flexible alternative data transfer option.

Pickup and Drop is composed of one or more source object Pickup operations followed by a single Drop operation on a target object. Pickup and Drop is initiated by the first Pickup and is terminated by a Drop or Cancel Drag operation. When Pickup and Drop is initiated, the mouse pointer is augmented with the system Pickup pointer icon.

The drag images seen in a standard direct manipulation operation are not displayed. As additional items are selected, they are added to the system Pickup set, and pickup emphasis is displayed for each item. The Pickup set is currently limited to a single source window or folder. While the operation is in progress, all other operations are valid with the exception of a standard direct manipulation operation.

Pickup and Drop is initiated by DrgLazyDrag in response to a WM_PICKUP message generated when the user presses Alt+mouse button 2 on a source object. As the pointer moves over a potential target, DM_DRAGOVER and DM_DRAGLEAVE messages are sent when the user presses a key indicating the intention to drop the object. The target emphasis is not displayed until the user attempts to drop the object. Each time items are added to the Pickup set in response to a WM_PICKUP message, DrgReallocDraginfo must be called to reallocate the DRAGINFO data structure. The Pickup and Drop operation is then re-initiated by another DrgLazyDrag call. DrgLazyDrag returns upon initialization for the operation. The pointing device remains active during the operation and may be used as if no drag operation were in effect. If the pointer is over a valid target when a Drop is invoked, a DM_DROP message is sent to the target, and a DM_DROPNOTIFY message posted to the source window.

DrgCancelLazyDrag is called to cancel the operation, and similarly posts a DM_DROPNOTIFY message to the source window, but with a target window handle of zero in the *mp2* parameter.

DrgLazyDrop can be used to programmatically invoke a drop operation; for example, from a menu choice.

DrgQueryDraginfoPtrFromHwnd and DrgQueryDraginfoPtrFromDragitem are called to query the DRAGINFO pointer at any time during the course of the operation.

DrgQueryDragStatus is called to determine whether a Pickup and Drop operation is currently in progress.

Data Structure Handling

Prior to initiating a Pickup and Drop operation (via DrgLazyDrag), DrgAllocDraginfo must be called to allocate a DRAGINFO data structure. As additional objects are added to the Pickup set, the DRAGINFO and DRAGITEM data structures must be reallocated using DrgReallocDraginfo. This function unconditionally frees the existing DRAGINFO data structure passed to it, reallocates a new DRAGINFO data structure, and returns the pointer to the new data structure. The Pickup and Drop operation is then re-initiated by another DrgLazyDrag call.

The DRAGIMAGE array is passed to DrgLazyDrag, so that compatibility with the drag operation is maintained. This allows the application to support Pickup and Drop, and standard drag operation with the same code. However, the drag images in the data structure are not used for display during Pickup and Drop, as the mouse pointer is augmented with a Pickup icon during the operation. As soon as DrgLazyDrag returns, the DRAGIMAGE array can be freed.

Message Handling

In the standard direct manipulation protocol, DrgDrag does not return until the drag set is dropped on a target window. Pickup and Drop is slightly different, and requires a change in the handling of a Drop. Because the operation is non-modal, DrgLazyDrag returns as soon as it has completed drag initialization and before a drop is performed. In the Pickup and Drop protocol, DM_DROPNOTIFY is posted to the source window as notification of a drop. The parameters of this message contain the pointer to the DRAGINFO data structure allocated by the source window and the handle of the target window. The source window should examine the *mp2* parameter to determine if the target window and the source window are the same; if not, the source should free the DRAGINFO upon receipt of this message. Where the target and source are the same, the target window frees DRAGINFO after completing the post-drop conversation. The implementation of Pickup and Drop does not affect any of the existing post-drop conversation messages.

The DM_DROPHELP message is not supported for the Pickup and Drop protocol, because help could be requested for any subject at any point during the operation. If the application is to provide Drop help, it must do so from a menu choice and explicitly code the support to be provided.

About Rendering Mechanisms

The following sections describe the standard rendering mechanisms used by various containers and applications for direct manipulation.

OS/2 File Rendering Mechanism

This rendering mechanism can be used by various containers, including file folders and trash cans. These containers allow objects to be dragged and dropped on white space in the container to accomplish a Move or Copy operation. They also can allow objects in the same or another container to be dragged and dropped on objects within the container to accomplish an operation.

Mechanism Name

The string for this rendering mechanism is `DRM_OS2FILE`.

Messages

The following messages are used by the `DRM_OS2FILE`:

- `DM_RENDER`

This message is sent by a target to a source to request a rendering for an object. When this message is received, the source determines if it understands the rendering mechanism and format selected by the target for the object. It also confirms that it allows the operation selected by the user for that object. The source must respond to this message before proceeding with the rendering operation.

- `DM_RENDERCOMPLETE`

This message is posted by a source to a target to notify the target that the rendering operation has been completed by the source, either successfully or unsuccessfully. The source can elect to let the target retry a successful or an unsuccessful operation. In this case, it should return to its state at the time of the drop for that object and indicate, in the message, that a retry is allowed.

Support for this message by a source is optional. If this message is not supported, then:

- The source must convey all necessary information to the target in order to allow it to handle the rendering operation.
- It must always indicate that native rendering is allowed when replying to a `DM_RENDER` message.

- `DM_ENDCONVERSATION`

This message is sent by a target to a source to notify the source that the rendering operation is complete and that the conversation is terminated. When this message is received, the entire drop operation for the object is complete. The source can now release any resources it had allocated to the drop and rendering operations. When the reply is received, the target can release the resources it had allocated to the operation.

Native Mechanism Actions

If the target understands the native rendering mechanism and format of the object, it may be possible to render the object without any involvement on the part of the source, provided the source has given the target sufficient information to do so. In order for the rendering to be performed by the target, the source must fill in, at a minimum, the *hstrContainerName* and *hstrSourceName* fields. The *hstrContainerName* field represents the subdirectory that the file indicated by *hstrSourceName* is in. For the target to do the rendering on its own, the true type of the object must be DTYP_OS2FILE. When these conditions are met, the target may proceed with the operation. When the operation is complete, the target must send a DM_ENDCONVERSATION message to the window indicated by *hwndItem* in the DRAGITEM data structure.

Preventing a Target from Rendering an Item

A source can prevent a target from doing the rendering operation on its own by not providing the source name for the object. This may be a necessary action for sources that implement some type of security, or that may not allow particular operations to be performed for an object move. When a source takes this course, it must fill in the *hstrSourceName* in the DRAGITEM data structure before replying to a DM_RENDER message. The target deletes the *hstrSourceName* string handle prior to freeing the DRAGINFO data structure, just as it would if the information had been passed to it at the time of the drop.

Requesting the Source to Render the Item

Whenever the conditions for a target to do the rendering operation without source participation are not met, the target must request the source to carry out the rendering by posting a DM_RENDER message to the source. Of course, the target can do this even if it is able to carry out the rendering mechanism on its own.

Allocating and Freeing a DRAGTRANSFER Data Structure

The data in a drag transfer message is carried in a DRAGTRANSFER data structure. DRAGTRANSFER data structures are allocated when the target calls DrgAllocDragtransfer.

When the conversation is completed, both the source and the target must call DrgFreeDragtransfer to free the shared memory. The target should do it immediately after sending a DM_ENDCONVERSATION message. The source should do it immediately after sending a DM_RENDERCOMPLETE message.

Operation Specifics

Regardless of the operation being performed, the target must fill in the *hstrRenderToName* field in the DRAGTRANSFER data structure before sending a DM_RENDER message. This is the fully qualified drive, path, and file name of the file that will contain the data when the rendering operation is complete. When the source has completed the operation, it must post a DM_RENDERCOMPLETE message to the target. The target then must complete the direct manipulation operation for that object by posting a DM_ENDCONVERSATION message to the source. Once the conversations for all of the objects involved in the drop are complete, the target can delete the string handles and free the DRAGINFO data structure.

Non-Native Mechanism Actions

The target may select the DRM_OS2FILE rendering mechanism when it is not the native rendering mechanism for an object, as long as the source supports it. In this case, the target must always request that the source carry out the rendering operation as described above. The

source should render the data in the requested format to the file specified by the *hstrRenderToName* field. If the requested operation is a Move, the source should take whatever action is necessary to remove its knowledge of the object as long as no information regarding the object was lost in the transfer.

Naming Conventions

The naming conventions for this rendering mechanism are as follows:

- **hstrContainerName**

Contains the fully qualified drive and path name for the source file, for example:

```
C:\
C:\MYSUBDIR\
A:\SUBDIR1\SUBDIR2\
\\NETWORK\SHARED\SUBDIRA\SUBDIRB\
```

- **hstrSourceName**

Contains the name of the source file or subdirectory, for example:

```
MYSOURCE.C
MYSOURCE.H
MYSOURCE IS A LONG FILE NAME
SUBDIR3
```

If you specify a subdirectory, the action is applied to all files in the subdirectory.

- **hstrRenderToName**

Contains the fully qualified file or subdirectory name that is to be used at the target, for example:

```
C:\MYSUBDIR\MYSOURCE.C
\\NETWORK\SHARED\SUBDIRA\SUBDIRB\MYSOURCE.H
C:\SUBDIR1\SUBDIR2\SUBDIR3
```

Types

Any type that is allowed as a *TYPE* extended attribute is allowed in the *hstrType* field of the DRAGITEM data structure. The type for a file can be obtained using DosQFileInfo; the type can be set by using DosSetFileInfo.

Print Rendering Mechanism

A common object that might be provided by a container is a printer. This object would allow objects to be dragged and dropped on it to accomplish a print operation.

Mechanism Name

The string for this rendering mechanism is DRM_PRINTOBJECT.

Messages

To support this rendering mechanism, a source must be able to receive and process a DM_PRINTOBJECT message. The target posts this message to the source. When the message is received, the source prints the current view of the object identified in the message to the printer. The second message parameter (of type PRINTDEST) gives all the parameters necessary to call DevPostDeviceModes and DevOpenDC.

Native Mechanism Actions

There are no native mechanism actions for this rendering mechanism, because the act of printing an object is considered a transform from the native rendering mechanism to the print mechanism.

Naming Conventions

None.

Dynamic Data Exchange (DDE) Rendering Mechanism

This rendering mechanism can be used by various containers and applications. The containers allow objects to be dragged and dropped on white space in the container to accomplish a Move or Copy operation. They can also allow objects in the same or another container to be dragged and dropped on objects within the container to accomplish some operation.

Mechanism Name

The string for this rendering mechanism is DRM_DDE.

Messages

To support this rendering mechanism, a source must be able to receive and process the following messages:

- WM_DDE_REQUEST

This message is posted by the target to the window indicated by the *hwndItem* field in the DRAGITEM data structure to request information regarding the object. Note that WM_DDE_INITIATE is not required because the target already has the handle of the window it wants to converse with. This message is sent for all Move and Copy operations.

- WM_DDE_ADVISE

This message is posted by the target to the window indicated by the *hwndItem* field in the DRAGITEM data structure order to maintain a *hot link* to the object.

- WM_DDE_UNADVISE

This message is posted by the target to the window indicated by the *hwndItem* field in the DRAGITEM data structure to terminate a hot link to the object.

- WM_DDE_TERMINATE

This message is posted by the target to the window indicated by the *hwndItem* field in the DRAGITEM data structure to terminate a conversation.

To support this rendering mechanism, a target must be able to receive and process the following messages:

- WM_DDE_DATA

This message is posted to the target by the source to deliver the requested information regarding the object.

- WM_DDE_ACK

This message is posted to the target by the source to acknowledge a WM_DDE_ADVISE or WM_DDE_UNADVISE message.

- WM_DDE_TERMINATE

This message is posted to the target by the source to end a conversation.

Native Mechanism Actions

Prior to establishing a DDE conversation, the target should determine the source-supported formats in which it wants to have the object rendered. It should register this format in the system atom table and use the resulting atom in the *usFormat* field of the DDESTRUCT used in the conversation.

The target should establish the DDE conversation by posting a WM_DDE_REQUEST message to the window indicated by the *hwndItem* field in the DRAGITEM data structure. The target acts as the client, and the source acts as the server in the conversation.

Operation Specifics

The following actions should be taken by the source, depending on the operation being performed:

Copy	Send the data to the target.
Move	Remove knowledge of the object after receiving confirmation that the target has successfully completed its portion of the rendering operation.

Non-Native Mechanism Actions

The target and source proceed in the same way, regardless of whether DDE was the native rendering mechanism or an alternate rendering mechanism.

Naming Conventions

The naming conventions for the DRM_DDE rendering mechanism follow:

- `hstrSourceName`
Contains the object name to be used in the DDE conversation.
- `hstrRMF`
The format portion of the list of ordered pairs in the format *<DRM_DDE,format>* identifies the formats supported by the source for the object. The non-standard DDE formats that these formats map to must be registered in the system atom table by both the source and the target.

Types

Any type that is allowed as a *.TYPE* extended attribute is allowed in the *hstrType* field of the DRAGITEM data structure.

Application-Defined Rendering Mechanisms

An application can choose to define a new rendering mechanism. However, if an application intends to provide renderings from this extended rendering mechanism to existing rendering mechanisms, it should publish enough information so that other application developers can use the new mechanism. An application must address several distinct areas of definition. These areas are described below, in general, and also are addressed under the definition for the system mechanisms.

Mechanism Name

The string name of the rendering mechanism should be defined by the application. This string name is specified in the mechanism/format pair of the DRAGITEM data structure.

Native Mechanism Actions

When both a source and target application store the data in the same native mechanism, a transform is not required. Instead, the native Move and Copy actions for that mechanism can be performed by the target. An application must completely define the proper procedure for performing that action. In the case of files, the native Move action is defined as a `DosMove` or `DosCopy/DosDelete`. The native Copy action is `DosCopy`. An application need not support all of the basic actions; it can choose to define additional native mechanism actions, indicated by the `DO_UNKNOWN` action in the DRAGINFO data structure.

Naming Conventions

An application that is defining a new mechanism must completely specify the naming conventions for objects rendered in that mechanism. This information typically includes both the name of the data and preceding information describing the exact location of the data. Any special rules concerning uppercase and lowercase or character sets to be used in naming also must be specified. The semantics for using these mechanism names, as well as an algorithm for deriving location information, also must be defined.

An application that is defining a new rendering mechanism must completely define the set of messages that a target and source application must support, and must specify the appropriate action to be taken for each message. The message IDs (above WM_USER) for the messages must be published.

Performance Considerations

If an application provides or defines transforms from the newly defined mechanism to existing mechanisms, performance information about the transform between mechanisms should be provided. This aids the application developer in choosing the appropriate transform when it encounters an application that transforms from an unknown native mechanism to several different known mechanisms.

Using Direct Manipulation

This section shows the sequence of function and message flows for a typical direct manipulation operation. It also describes the activities that must be performed by the applications during direct manipulation.

Note: Most of the sample code in this section is part of a complete program illustrated in "Sample Code for Direct Manipulation".

Allocating Memory for the Drag Operation

To prepare for the drag operation, the source must invoke `DrgAllocDraginfo` to allocate memory for the DRAGINFO data structure. `DrgAllocDraginfo` initializes the DRAGINFO data structure as follows:

<i>cbDraginfo</i>	The size, in bytes, of the entire DRAGINFO data structure, including the DRAGITEM array
<i>cbDragitem</i>	The size, in bytes, of each DRAGITEM data structure
<i>usOperation</i>	DO_DEFAULT
<i>xDrop</i> and <i>yDrop</i>	The current mouse-pointer location, in desktop coordinates
<i>cditem</i>	The count of objects being dragged, as specified in <code>DrgAllocDraginfo</code> .

Initializing DRAGITEM Data Structure

After allocating memory for the DRAGINFO data structure, the source initializes a DRAGITEM data structure, as appropriate, for each of the objects to be dragged. This is accomplished either by using `DrgSetDragitem` or by obtaining a pointer to each DRAGITEM data structure with `DrgQueryDragitemPtr`, and initializing it directly.

The first step the source takes to initialize the DRAGITEM data structure is to create the appropriate drag string handles. String handles must be created for:

- Object type
 - Supported rendering mechanisms and formats for the object
 - Suggested name of the object at the target
 - Name of the container holding the object (whether a container or folder)
 - Name of the object at the source when the source allows the target to carry out the operation for the object
-

Type

To directly manipulate an object, both the source and the target must support the object *type*, which describes the format of the object. For example, the input to a C compiler could have the type *Plain Text* (DRT_TEXT). The *hstrType* field in the DRAGITEM data structure conveys this information for each object being dragged. The type is represented by a string handle. The target should check to see if it supports the type before allowing the user to drop the object.

Several DTYP_* constants are defined as *notational conveniences* for common types of data. An application can extend these types by defining its own character strings and then creating string handles for them using DrgAddStrHandle.

True Type

The *true type* of an object is the type that most accurately describes the object. For example, the input to a C compiler could have the type *Plain Text* (DRT_TEXT), but would be more accurately described as *C Code* (DRT_C). *C Code* would be the true type of this object. Multiple types can be conveyed by using a comma to separate strings. The following figure shows the format to use to convey multiple types:

```
"type,type..."
```

The true type should appear first in the list of types, so the type string for the example object would be "C Code, Plain Text".

Rendering Mechanism and Format

The rendering mechanism and format are passed as a string handle in the DRAGITEM data structure. The string handle must be created using DrgAddStrHandle. The following figure shows the string handle format:

```
"elem {,elem,elem...}"
```

where *elem* is an ordered pair in the form:

```
"<mechanism,format>"
```

or a cross product in the form:

```
"(mechanism{,mechanism...}) X (format{,format...})"
```

Multiple cross products are permitted in a single rendering mechanism and format string handle, as are combinations of ordered pairs and cross products. When cross-product notation is used, the rendering mechanism is the left operand. When ordered-pair notation is used, the rendering mechanism is the left element in the ordered pair.

Several constants are defined for common rendering mechanisms and formats. For example, the rendering mechanism and format for a:

- C source file might be "<DRM_OS2FILE,CF_OEMTEXT>"
- Spreadsheet file might be "<DRM_OS2FILE,CF_SYLK>"

An application can extend these by defining its own "<mechanism,format>" strings and creating string handles for these using DrgAddStrHandle. For example, if an application understands and can generate an LU 6.2 data stream, it can define its own rendering format, "DRF_LU62", and use it in direct manipulation operations. If an application wishes to use its own rendering mechanisms or formats to communicate with other applications, it should publish the protocol for the mechanisms, the format of the data streams, or both.

Native Rendering Mechanism and Format

The native rendering mechanism and format of the object is the mechanism that most naturally conveys the data and its current format. For example, the native rendering mechanism and format for a:

- C source file might be "<DRM_OS2FILE,CF_OEMTEXT>"
- Spreadsheet file might be "<DRM_OS2FILE,CF_SYLK>"

In some direct-manipulation operations, it might be possible for the target to carry out the necessary action on the source object without the source's participation. However, this is possible only when the target supports both the true type and the native rendering mechanism and format of the object. Even when the target is not performing the necessary action on the source object, it is still important to know the native rendering mechanism and format. In determining the rendering mechanism and format to be used in the data exchange after the drop, the target might select the native format because, generally, performance is better when the native rendering mechanism and format are used.

The native rendering mechanism and format are conveyed to the target by making it the first ordered pair, or the first ordered pair to result from a cross product, in the list of rendering mechanisms and formats passed in the DRAGINFO data structure.

Suggested Name at Target

When dragging an object, for example, a file, from one container to another, it is important to know the name the object should have at the target. This may or may not be the same name it had at the source. This name enables the target to check if another object with the same name already exists at the target and to take the appropriate action. For example, a target container might not allow the user to drop the object if an object by that same name already exists at the target.

Container Name

Sometimes it is necessary for a target container to know the name of the source container. This name could carry some location information. For example, the default operation when dragging objects between containers is a Move. However, in the case of file folders on different drives, this default would be changed to a Copy operation. Thus, a file folder would fill this field with the drive and path information for a file, for example, A:\SUBDIR1\SUBDIR2\ . A database container, on the other hand, might fill this field with the fully qualified OS/2 file name of the database.

Source Name

In some direct-manipulation operations, it is possible for the target to perform the necessary action on the source object without the source's participation. If the source allows this, the target name should be filled in with the name of the source object. For example, a file folder would put the name of the source file into this field, such as AUTOEXEC.BAT. A database manager, on the other hand, might fill this field with some location information so the target could find a particular record or field within the database.

Sample Code for Initializing DRAGITEM Data Structure

The following code fragment shows how to initialize the DRAGITEM array:

```
/* *****  
/*  Get our current directory for the container name.  */  
/* *****  
dirlen          = CCHMAXPATH-1;
```

```

DosQueryCurrentDir(0, szDir, &dirLen);
sprintf(szContainer, "\\%s\\", szDir);
hstrContainer      = DrgAddStrHandle(szContainer);
Dragitem.hwndItem  = hListWnd;
Dragitem.hstrType  = hstrType;
Dragitem.hstrRMF   = hstrRMF;
Dragitem.hstrContainerName = hstrContainer;
Dragitem.fsControl = 0;
Dragitem.fsSupportedOps = DO_COPYABLE | DO_MOVEABLE;
Dragitem.hstrSourceName = DrgAddStrHandle (szBuffer);
Dragitem.hstrTargetName = Dragitem.hstrSourceName;
Dragitem.ulItemID   = index;

/*****
/*  Set info, prepare for drag.
*****/
DrgSetDragitem(pSourceDraginfo,
               &Dragitem,
               sizeof(DRAGITEM),
               0);

```

Initializing DRAGIMAGE Data Structure

As part of the preparation for the actual drag, an application initializes a DRAGIMAGE data structure. The following sample code shows how to initialize the DRAGIMAGE data structure:

```

/*****
/*  Initialize the drag image.
*****/
dimg.cb      = sizeof (DRAGIMAGE);
dimg.hImage  = WinQuerySysPointer (HWND_DESKTOP, SPTR_FILE, FALSE);
dimg.fl      = DRG_ICON | DRG_TRANSPARENT;
dimg.cxOffset = 0;
dimg.cyOffset = 0;

```

Starting the Drag Operation

Once initialization is complete, the source object calls DrgDrag to start the direct manipulation operation. The following sample code shows how to start the drag operation:

```

/*****
/*  Start drag operation.
*****/
DrgDrag(hFrameWnd,
        pSourceDraginfo,
        &dimg,
        1L,
        VK_BUTTON2,
        NULL);

```

Responding to the DM_DRAGOVER Message

The DM_DRAGOVER message is sent to a target whenever the user drags the pointer into the window. To assess whether a drop can be accepted, the target must use DrgAccessDraginfo to get access to the DRAGINFO data structure. It then determines whether a drop can be accepted for each object. The object must meet the following minimum requirements to exchange data:

- The source and target must share knowledge of at least one common type for the object. The target can make this determination by using DrgVerifyTypeSet or DrgVerifyType.
- The source and target must share at least one common rendering mechanism and format for that type object. The target can make this determination by using DrgVerifyRMF.

DOR_DROP, DOR_NODROP, DOR_NODROPOP, and DOR_NEVERDROP are the four possible responses available to the target when it receives a DM_DRAGOVER message. The target sends these values to the window handle specified in the DRAGINFO data structure.

The following sample code shows how the target determines its response to the DM_DRAGOVER message:

```

/*****
/*  Someone's dragging an object over us.
*****/
case DM_DRAGOVER:
    dragInfo = (PDRAGINFO)mp1;

    /* Get access to the DRAGINFO data structure */
    DrgAccessDraginfo(dragInfo);

    /* Can we accept this drop? */
    switch (dragInfo->usOperation)
    {

        /* Return DOR_NODROPOP if current operation */
        /* is link or unknown */
        case DO_UNKNOWN:
            DrgFreeDraginfo(dragInfo);
            return (MRFROM2SHORT(DOR_NODROPOP,0));
            break;

        /* Our default operation is Move */
        case DO_DEFAULT:
            dragItem = DrgQueryDragitemPtr(dragInfo,0);
            ulBytes = DrgQueryStrName(dragItem->hstrContainerName,
                                    sizeof(szDir),
                                    szDir);

            if (!ulBytes)
                return (MRFROM2SHORT(DOR_NODROPOP,0));
            else
                usOp = DO_MOVE;
            break;

        /* Do the requested specific operation */
        case DO_MOVE:
        case DO_COPY:
            usOp = dragInfo->usOperation;
            break;
    }

    usIndicator = DOR_DROP;
    cItems = DrgQueryDragitemCount(dragInfo);

    /* Now, we need to look at each item in turn */
    for (i = 0; i < cItems; i++)
    {
        dragItem = DrgQueryDragitemPtr(dragInfo, i);

        /* Make sure we can move for a Move request */
        /* or copy for a Copy */
        if (((dragItem->fsSupportedOps & DO_COPYABLE) &&
            (usOp == (USHORT)DO_COPY)) ||
            ((dragItem->fsSupportedOps & DO_MOVEABLE) &&
            (usOp == (USHORT)DO_MOVE)))
        {
            /* Check the rendering format */
            if (DrgVerifyRMF(dragItem, "DRM_OS2FILE", "DRF_UNKNOWN"))
                usIndicator = DOR_DROP;
            else
                usIndicator = DOR_NEVERDROP;
        }
    }
}

```

```

    }
    else
        usIndicator = DOR_NODROPOP;
}

/* Release the draginfo data structure */
DrgFreeDraginfo(dragInfo);

return (MRFROM2SHORT(usIndicator, usOp));
break;

```

Providing Target Emphasis

The target should provide target emphasis so the user knows exactly where the drop occurs or, if the drop is not allowed, the boundaries of the region where the drop is not allowed.

A container window should emphasize a target object by drawing a thin, black rectangle around it. The application should use DrgGetPS and DrgReleasePS to obtain the presentation space in which to draw target emphasis.

Providing Customized Images

The target can provide a customized pointer to be displayed while it is the target of the drop by calling DrgSetDragPointer after it starts processing the DM_DRAGOVER message but before it sends a response. It also can provide a customized image (icon, bit map, and so forth) to be displayed while it is the target by calling DrgSetDragImage. This capability may be used by a target to provide additional visible feedback to the user. The pointer is reverted to the default when it is moved to a new target.

Responding to the DM_DRAGLEAVE Message

DM_DRAGLEAVE is sent whenever the DM_DRAGOVER message is sent to a window, and the pointer is moved outside the bounds of that window. If the target or an object in the window had been emphasized as a target, it should be de-emphasized.

Container windows monitor the position of the pointer on DM_DRAGOVER messages and simulate the DM_DRAGLEAVE message when the pointer moves on or off a contained object.

A DM_DRAGLEAVE message is not sent if the user drops the objects being dragged within the window. Therefore, when DM_DROP is received, the application de-emphasizes any target that was emphasized as a valid target.

If the user drags the pointer outside the target window, resulting in a new target, a DM_DRAGLEAVE message is sent to the former target. The receiver of a DM_DRAGLEAVE message should use it to de-emphasize the target, thus providing the user with visible feedback that this is no longer the target.

Responding to the DM_DROP Message

When the user drops the objects, a DM_DROP message is sent to the target, providing it with the information necessary to process the objects that were dropped. The target application uses the information provided to exchange data with the source. The target is responsible for establishing the appropriate conversations, and the source must cooperate in establishing the necessary conversations to achieve the actual data exchange. After completing the direct manipulation operation, including the post-drop conversation with the source, the target uses DrgDeleteStrHandle or DrgDeleteDraginfoStrHandles to delete the string handles in the DRAGINFO data structure, and DrgFreeDraginfo to release the storage. The target should immediately remove any target emphasis. The data transfers must not be done before responding to the DM_DROP message.

The following sample code shows how a target processes an object that has been dropped on it. This code fragment is part of a complete program which is illustrated in "Sample Code for Direct Manipulation".

```

/* Drop the object on us (receive the object) */
case DM_DROP:

    /* Get access to the DRAGINFO data structure */
    DrgAccessDraginfo(dragInfo);

    /* Can we accept this drop? */
    switch (dragInfo->usOperation)
    {

        /* Return DOR_NODROPOP if current */
        /* operation is link or unknown. */
        case DO_UNKNOWN:
            DrgFreeDraginfo(dragInfo);
            return (MRFROM2SHORT (DOR_NODROPOP, 0));
            break;

        /* Our default operation is Move */
        case DO_DEFAULT:
            dragItem = DrgQueryDragitemPtr(dragInfo, 0);
            ulBytes = DrgQueryStrName(dragItem->hstrContainerName,
                                     sizeof(szDir),
                                     szDir);

            if(!ulBytes)
                return (MRFROM2SHORT (DOR_NODROPOP, 0));
            usOp = (USHORT)DO_MOVE;
            break;

        /* Do the requested specific operation */
        case DO_MOVE:
        case DO_COPY:
            usOp = dragInfo->usOperation;
            break;
    }

    usIndicator = DOR_DROP;
    cItems = DrgQueryDragitemCount(dragInfo);

    /* Now, we need to look at each item in turn */
    for (i = 0; i < cItems; i++)
    {
        dragItem = DrgQueryDragitemPtr(dragInfo, i);

        /* Make sure we can move for a Move request */
        /* or copy for a Copy */
        if (((dragItem->fsSupportedOps & DO_COPYABLE) &&
            (usOp == (USHORT)DO_COPY)) ||
            ((dragItem->fsSupportedOps & DO_MOVEABLE) &&
            (usOp == (USHORT)DO_MOVE)))
        {
            /* Check the rendering format */
            if (DrgVerifyRMF(dragItem, "DRM_OS2FILE", "DRF_UNKNOWN"))
                usIndicator = DOR_DROP;
            else
                usIndicator = DOR_NEVERDROP;
        }
        else
            usIndicator = DOR_NODROPOP;

        /*****
        /* This is where we would actually move or copy the file,
        /* but we just display the name instead.
        *****/
        DrgQueryStrName(dragItem->hstrSourceName, 255, szBuffer);
        WinMessageBox(HWND_DESKTOP,
                     HWND_DESKTOP,
                     szBuffer,
                     "Dropped",
                     0,
                     MB_OK);
    }

    /* Release the draginfo data structure */

```

```

DrgFreeDraginfo(dragInfo);

return (MRFROM2SHORT(usIndicator, usOp));
break;

```

Exchanging Data

Direct manipulation offers various ways for source and target applications to exchange data. To accomplish the exchange, a separate conversation must be established to transfer each data object from the source to the target. The target must inform the source about the rendering mechanism it is using and the format in which the data is to be exchanged. The target can establish the conversations to run in parallel, or it can initiate the conversations in a serial fashion.

The target determines which rendering mechanism and format to use in the following manner:

1. Uses the native rendering mechanism and format whenever possible.

This rendering conveys all information about the data. A target can determine if it supports the native rendering mechanism and format by using the following functions:

- DrgVerifyNativeRMF
- DrgQueryNativeRMFLen
- DrgQueryNativeRMF

Even if it can use the native rendering mechanism and format supported by the source, the target can elect to exchange the data in a rendering mechanism and format that conveys less information about the object.

2. Uses the next best rendering mechanism and format.

This is especially good for a Copy operation, because the user does not lose data about the object as occurs when the object is moved.

The target can determine the next best rendering mechanism and format to use through repeated calls to DrgVerifyRMF. The calls are made starting with the most desirable rendering mechanism and format pair and progressing to the least desirable pair. Once a pair that the source supports has been found, the target can exchange the data.

The following sample code shows how the target checks the rendering mechanism and format:

```

/* Now, we need to look at each item in turn */
for (i = 0; i < cItems; i++)
{
    dragItem = DrgQueryDragitemPtr(dragInfo, i);

    /* Make sure we can move for a Move request */
    /* or copy for a Copy */
    if (((dragItem->fsSupportedOps & DO_COPYABLE) &&
        (usOp == (USHORT)DO_COPY)) ||
        ((dragItem->fsSupportedOps & DO_MOVEABLE) &&
        (usOp == (USHORT)DO_MOVE)))
    {
        /* Check the rendering format */
        if (DrgVerifyRMF(dragItem, "DRM_OS2FILE", "DRF_UNKNOWN"))
            usIndicator = DOR_DROP;
        else
            usIndicator = DOR_NEVERDROP;
    }
    else
        usIndicator = DOR_NODROPOP;
}

```

Performance Considerations

When context information about an object might be lost because of using a less-desirable rendering mechanism and format, the target can elect to pick a common mechanism and format that achieves the best performance. This is done the same way that the next best rendering mechanism and format is selected, proceeding from the best-performing rendering to the worst.

Regardless of the rendering mechanism used, the target might need to prepare the source for the rendering of the object. This is necessary when the source needs to create a window in order to handle the conversation. This preparation is done by sending a DM_RENDERPREPARE message to the *hwndSource* window in the DRAGINFO data structure. This message need be sent only when the DC_PREPARE flag is on in the *fsControl* field of the DRAGITEM data structure. When the source receives this message, it performs any necessary preparation for the rendering and fills in the *hwndItem* field in the DRAGITEM data structure, thereby allowing the target to establish conversation with that window.

Using Pickup and Drop

The following sample code shows a Pickup and Drop operation after the user has selected an object and pressed mouse button 2 while holding down the Pickup and Drop augmentation key (Alt):

```
#define INCL_WINSTDDRAG #include <os2.h>

PDRAGINFO pdinfo;          /* Pointer to a DRAGINFO data structure */
HWND hwndSource;           /* Handle of the Source window */
DRAGITEM ditem;            /* DRAGITEM data structure */
PDRAGIMAGE pdimg;          /* Pointer to DRAGIMAGE data structure */
HBITMAP hbm;               /* Bit-map handle passed to DrgLazyDrag */
APIRET rc;                 /* Allocation memory return code */

case WM_PICKUP:

/* *****
/* Initialize the DRAGITEM data structure.
/* *****
ditem.hwndItem=hwndSource;    /* Handle of the source window */
ditem.ulItemID=ID_ITEM;      /* App. defined id of item */

ditem.hstrType                = DrgAddStrHandle("DRT_TEXT"); /* Text item */
ditem.hstrRMF                 = DrgAddStrHandle("<DRM_OS2FILE,DRF_TEXT>");
ditem.hstrContainerName       = DrgAddStrHandle("C:\\");
ditem.hstrSourceName          = DrgAddStrHandle("C:\\\\CONFIG.SYS");
ditem.hstrTargetName          = DrgAddStrHandle("C:\\OS2\\CONFIG.SYS");

ditem.cxOffset                = 0; /* Offset of the origin of the image */
ditem.cyOffset                = 0; /* from the pointer hotspot */
ditem.fsControl                = 0; /* Source item control flags */
ditem.fsSupportedOps          = 0;

/* *****
/* Create the DRAGINFO data structures
/* *****
pdinfo=DrgAllocDraginfo(1);

/* Return FALSE if initialization fails */
if(!pdinfo) return FALSE;

/* *****
/* Initialize the DRAGIMAGE data structure.
/* *****
rc=DosAllocMem((PPVOID)&pdimg, /* Allocate memory */
               sizeof(DRAGIMAGE),
               (ULONG)PAG_COMMIT |
               PAG_READ |
               PAG_WRITE);

pdimg->cb=sizeof(DRAGIMAGE); /* Size of the dragimage structure */
pdimg->cptl=0;                /* Image is not a polygon */
pdimg->hImage=hbm;           /* Handle of image to display */
pdimg->szlStretch.cx=20L;    /* Size to stretch icon or bit map */
pdimg->fl=DRG_BITMAP |      /* Flags passed to DrgLazyDrag */
        DRG_STRETCH;
```

```

pdimg->cxOffset=0;                /* Offset of the origin of image */
pdimg->cyOffset=0;                /* from the pointer hotspot */

/*****
/* Set the DRAGITEM data structure.
*****/
DrgSetDragitem(pdinfo,&ditem,(ULONG)sizeof(ditem),0);

/*****
/* Begin the Lazy Drag operation.
*****/
if (DrgLazyDrag(hwndSource,      /* Source of the drag */
                pdinfo,          /* Pointer to the DRAGINFO */
                pdimg,           /* DRAGIMAGE array */
                1,               /* Size of the DRAGIMAGE */
                NULL))           /* Reserved */
{
    DosFreeMem(pdimg);           /* Free DRAGIMAGE if successful */
}

```

Graphical User Interface Support for Direct Manipulation

This section describes the support the direct manipulation provides for graphical user interfaces (GUIs). Except where noted, this support conforms to the guidelines in the *SAA CUA Advanced Interface Design Reference*.

Keyboard Augmentation

A direct manipulation operation begins in a *default state*, which means that, when the user drops objects on a target, the target is informed that it should perform its default operation. The target is responsible for defining its default operation. For a container window, the default should be a Move operation, if it is supported. The default for a device, such as a printer, should be a Copy operation.

As the user drags the object, the default operation can be overridden by pressing and holding one of the following augmentation keys:

Ctrl	Changes the operation to a Copy
Shift	Changes the operation to a Move
Ctrl+Shift	Changes the operation to a Link.

The last key pressed and held at the time of the drop determines the operation to be performed. The target can determine the defined augmentation key that was pressed at the time of the drop by inspecting the *usOperation* field of the DRAGINFO data structure.

A target can define additional augmentation keys for its own use. In this case, *usOperation* would indicate that the operation is unknown, and the target needs to use WinGetKeyState to determine the actual augmentation key that was used.

As the user presses augmentation keys, the pointer currently being displayed is modified to provide the user with a visible cue as to the type of operation being performed.

Sample Code for Direct Manipulation

This section illustrates a complete sample program for the drag portion of a drag-and-drop operation. Several parts of this program are explained in "Using Direct Manipulation".

Source Application Sample Code

The source application includes the following files:

- Dragfrom.C
- Dragfrom.H
- Dragfrom.DEF
- Dragfrom.LNK
- Dragfrom.MAK

The following sample shows the source application code:

```
=====
DRAGFROM.C
=====

/*****
/*  DRAGFROM.C - Drag source program
/*
/*  This program displays a list of files in the current directory.
/*  Drag any file name to EPM, and drop, and the file will be
/*  displayed in the editor.
*****/
#define INCL_DOSFILEMGR
#define INCL_WIN
#define INCL_WINSTDDRAG
#define INCL_WINLISTBOXES
#define INCL_WINWINDOWMGR

#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "dragfrom.h"

/*****
/*  Global variables.
*****/
HAB      hab;
char      szFormats[] = "<DRM_OS2FILE, DRF_UNKNOWN>";
char      szFileNames[50][CCHMAXPATH];
HWND      hFrameWnd;
HWND      hListWnd;
PFNWP     SysWndProc;
PFNWP     ListWndProc;
HPOINTER  hptrFile;

/*****
/*  Function prototypes.
*****/
MRESULT EXPENTRY LocalWndProc(HWND, ULONG, MPARAM, MPARAM);
MRESULT EXPENTRY LocalListProc(HWND, ULONG, MPARAM, MPARAM);
BOOL DoDrag(void);
void LoadList(void);

/*****
/*  Main() - program entry point.
*****/
int main(void)
{
    FRAMECDATA fcd;
    HMQ        hmq;
    QMSG       qmsg;

    if (!(hab = WinInitialize (0)))
        return FALSE;

    hmq = WinCreateMsgQueue (hab, 0);

    if (!hmq)
    {
        WinTerminate(hab);
        return FALSE;
    }
}
```

```

    }

/*****
/*  Setup the frame control data for the frame window.          */
*****/
fcd.cb = sizeof(FRAMECDATA);
fcd.flCreateFlags = FCF_TITLEBAR      |
                   FCF_SYSMENU       |
                   FCF_SIZEBORDER    |
                   FCF_SHELLPOSITION |
                   FCF_MINMAX        |
                   FCF_TASKLIST;

fcd.hmodResources = NULLHANDLE;

/*****
/*  Set our resource key (so PM can find menus, icons, etc).    */
*****/
fcd.idResources = DRAGFROM;

/*****
/*  Create the frame - it will hold the list box.              */
*****/
hFrameWnd = WinCreateWindow(HWND_DESKTOP,
                           WC_FRAME,
                           "Drag Source",
                           0, 0, 0, 0, 0,
                           NULLHANDLE,
                           HWND_TOP,
                           DRAGFROM,
                           &fcd,
                           NULL);

/*****
/*  Verify that the frame was created; otherwise, stop.         */
*****/
if (!hFrameWnd)
    return FALSE;

/*****
/*  Set an icon for the frame window.                           */
*****/
WinSendMsg(hFrameWnd,
           WM_SETICON,
           (MPARAM)WinQuerySysPointer(HWND_DESKTOP,
                                       SPTR_FOLDER,
                                       FALSE),
           NULL);

/*****
/*  Create a list window child - we will list files in it.      */
*****/
hListWnd = WinCreateWindow(hFrameWnd,
                           WC_LISTBOX,
                           NULL,
                           0, 0, 0, 0, 0,
                           hFrameWnd,
                           HWND_BOTTOM,
                           FID_CLIENT,
                           NULL,
                           NULL);

/*****
/*  We must intercept the frame window's messages.             */
/*  We save the return value (the current WndProc),             */
/*  so we can pass it all the other messages the frame gets.   */
*****/
SysWndProc = WinSubclassWindow(hFrameWnd, (PFNWP)LocalWndProc);
ListWndProc = WinSubclassWindow(hListWnd, (PFNWP)LocalListProc);

WinShowWindow(hFrameWnd, TRUE);
WinPostMsg(hFrameWnd, WM_LOAD_LIST, 0, 0);

/*****
/*  Main message loop.                                          */
*****/
while (WinGetMsg(hab, &qmsg, 0L, 0, 0))
    WinDispatchMsg(hab, &qmsg);

```

```

WinDestroyWindow (hFrameWnd);
WinDestroyMsgQueue (hmq);
WinTerminate (hab);
}

/*****
/* LocalWndProc() - intercepts frame window messages. */
*****/
MRESULT EXPENTRY LocalWndProc (HWND hwnd,ULONG msg,MPARAM mp1,MPARAM mp2)
{
    switch (msg)
    {
        /* Post a message to fill the list box */
        case WM_LOAD_LIST:
            LoadList();
            break;

        case WM_DESTROY:
            WinDestroyPointer (hptrFile);
            break;

        case WM_STARTDRAG:
            DoDrag();
            break;

        default:
            return (*SysWndProc)(hwnd, msg, mp1, mp2);
            break;
    }
    return FALSE;
}

/*****
/* LocalListProc() - List box subclassing */
/* (all we care about is starting a drag). */
*****/
MRESULT EXPENTRY LocalListProc(HWND hwnd,
                                ULONG msg,
                                MPARAM mp1,
                                MPARAM mp2)
{
    if (msg == WM_BUTTON2DOWN)
    {
        WinPostMsg(hFrameWnd, WM_STARTDRAG, mp1, 0);
        return (MRESULT)FALSE;
    }
    else
        return (*ListWndProc)(hwnd, msg, mp1, mp2);
}

/*****
/* DoDrag() - the actual drag function. */
*****/
BOOL DoDrag ()
{
    char                szBuffer[CCHMAXPATH];
    char                szDir[256];
    SHORT               index, len;
    HWND                hTargetWnd;
    LHANDLE              hImage;

    DRAGITEM             Dragitem;
    HSTR                 hstrType, hstrRMF, hstrContainer;
    CHAR                szItemName[64];
    CHAR                szContainer[CCHMAXPATH];
    PDRAGINFO            pSourceDraginfo;
    DRAGIMAGE            dimg;
    ULONG               dirlen;

    /* Get the file name from the listbox. */
    index = WinQueryLboxSelectedItem(hListWnd);
    len = WinQueryLboxItemTextLength(hListWnd, index);
    WinQueryLboxItemText(hListWnd,
                        index,
                        szBuffer,
                        len);

```

```

    szBuffer[len] = '\\0';

/*****
/*  Allocate the DRAGINFO data structure.
*****/
    pSourceDraginfo = DrgAllocDraginfo(1);

/*****
/*  Define file type as unknown.
*****/
    hstrType = DrgAddStrHandle (DRT_UNKNOWN);
    hstrRMF = DrgAddStrHandle (szFormats);          /* OS2file unknown */

/*****
/*  Get our current directory for the container name.
*****/
    dirlen = CCHMAXPATH-1;
    DosQueryCurrentDir(0, szDir, &dirlen);
    sprintf(szContainer, "\\%s\\", szDir);
    hstrContainer = DrgAddStrHandle(szContainer);
    Dragitem.hwndItem = hListWnd;
    Dragitem.hstrType = hstrType;

    Dragitem.hstrRMF = hstrRMF;
    Dragitem.hstrContainerName = hstrContainer;
    Dragitem.fsControl = 0;
    Dragitem.fsSupportedOps = DO_COPYABLE | DO_MOVEABLE;
    Dragitem.hstrSourceName = DrgAddStrHandle (szBuffer);
    Dragitem.hstrTargetName = Dragitem.hstrSourceName;
    Dragitem.ulItemID = index;

/*****
/*  Set info, prepare for drag.
*****/
    DrgSetDragitem(pSourceDraginfo,
        &Dragitem,
        sizeof(DRAGITEM),
        0);

/*****
/*  Initialize the drag image.
*****/
    dimg.cb = sizeof (DRAGIMAGE);
    dimg.hImage = WinQuerySysPointer (HWND_DESKTOP, SPTR_FILE, FALSE);
    dimg.fl = DRG_ICON | DRG_TRANSPARENT;
    dimg.cxOffset = 0;
    dimg.cyOffset = 0;

    pSourceDraginfo->hwndSource = hFrameWnd;

/*****
/*  Start drag operation.
*****/
    DrgDrag(hFrameWnd,
        pSourceDraginfo,
        &dimg,
        1L,
        VK_BUTTON2,
        NULL);

    return TRUE;
}

/*****
/*  LoadList().
*****/
void LoadList(void)
{
    char szDir[CCHMAXPATH];
    FILEFINDBUF3 ffbFile;
    HDIR hDir;
    int rc, x;
    ULONG dirlen;
    ULONG count;

/*****
/*  We use a DosFindFirst/DosFindNext loop to fill the list box.
*****/
    hDir = HDIR_CREATE;

```



```

count = 1;
rc = DosFindFirst("*.*",
                  &hDir,
                  0,
                  &ffbFile,
                  sizeof(FILEFINDBUF3),
                  &count,
                  FIL_STANDARD);

x = 0;
do
{
    sprintf(szFileNames[x], "%s", ffbFile.achName);

    WinPostMsg(hListWnd,
               LM_INSERTITEM,
               MPFROMSHORT(LIT_END),
               szFileNames[x]);

    count = 1;
    x++;

    rc = DosFindNext(hDir,
                     &ffbFile,
                     sizeof(FILEFINDBUF3),
                     &count);
}
while (count && (x < 50));

DosFindClose(hDir);
}

```

=====

DRAGFROM.H

=====

```

#define DRAGFROM      100
#define WM_STARTDRAG  WM_USER+100
#define WM_LOAD_LIST  WM_USER+110

```

=====

DRAGFROM.DEF

=====

```

NAME                DRAGFROM WINDOWAPI

```

PROTMODE

HEAPSIZE 8192

STACKSIZE 32768

```

EXPORTS              LocalWndProc
                     LocalListProc

```

=====

DRAGFROM.LNK

=====

```

dragfrom.obj
dragfrom.exe
dragfrom.map
dragfrom.def

```

=====

DRAGFROM.MAK

=====

```

CC      = gcc /c /Ge /Gd- /Se /Re /ss /Gm+

```

```

LINK    = link386

```

```

HEADERS = dragfrom.h

```

```

#-----

```

```

# A list of all of the object files.

```

```

#-----

```

```

ALL_OBJ1 = dragfrom.obj

```

```

all: dragfrom.exe

```

```

dragfrom.obj: dragfrom.c $(HEADERS)

```

```

dragfrom.exe: $(ALL_OBJ1) dragfrom.def dragfrom.lnk
               $(LINK) @dragfrom.lnk

```

Target Application Sample Code

The target application includes the following files:

- Target.C
- Target.RC
- Target.H
- Target.DEF
- Target.LNK

The following sample shows the target application code:

```
=====
TARGET.C
=====
#define INCL_WIN
#define INCL_GPI

#include <os2.h>
#include "target.h"

#pragma linkage (main,optlink)
INT main(VOID);

/*****
/* Main() - program entry point.
/* This program accepts drops from EPM.
*****/
MRESULT EXPENTRY LocalWndProc(HWND, ULONG, MPARAM, MPARAM);

HAB hab;
HWND hFrameWnd;
PFNWP SysWndProc;

INT main (VOID)
{
    HMQ hmq;
    HPOINTER hPtr;
    FRAMECDATA fcd;
    QMSG qmsg;

    if (!(hab = WinInitialize(0)))
        return FALSE;

    if (!(hmq = WinCreateMsgQueue(hab, 0)))
        return FALSE;

    /*****
    /* Set up the frame control data for the frame window.
    *****/
    fcd.cb = sizeof(FRAMECDATA);
    fcd.flCreateFlags = FCF_TITLEBAR |
                      FCF_SYSMENU |
                      FCF_SIZEBORDER |
                      FCF_SHELLPOSITION |
                      FCF_MINMAX |
                      FCF_TASKLIST;
    fcd.hmodResources = NULLHANDLE;
    fcd.idResources = 0;

    /*****
    /* Create the frame window.
    *****/
    hFrameWnd = WinCreateWindow(HWND_DESKTOP,
                                WC_FRAME,
                                "Target",
                                0,
                                0, 0, 0, 0,
                                NULLHANDLE,
                                HWND_TOP,
                                0,
```

```

        &fcd,
        NULL);

/*****
/* Verify that the frame was created; otherwise, stop. */
*****/
if (!hFrameWnd)
    return FALSE;
hPtr = WinLoadPointer(HWND_DESKTOP,
                     NULLHANDLE,
                     TRASHCAN);

/*****
/* Set an icon for the frame window. */
*****/
WinSendMessage(hFrameWnd,
               WM_SETICON,
               (MPARAM)hPtr,
               NULL);

/*****
/* We must intercept the frame window's messages */
/* (to capture any input from the container control). */
/* We save the return value (the current WndProc), */
/* so we can pass it all the other messages the frame gets. */
*****/
SysWndProc = WinSubclassWindow(hFrameWnd, (PFNWP)LocalWndProc);

WinShowWindow(hFrameWnd, TRUE);

/*****
/* Standard PM message loop - get it, dispatch it. */
*****/
while (WinGetMsg(hab, &qmsg, NULLHANDLE, 0, 0))
    WinDispatchMsg(hab, &qmsg);

/*****
/* Clean up on the way out. */
*****/
WinDestroyMsgQueue(hmq);
WinTerminate(hab);

return TRUE;
}

/*****
/* LocalWndProc() - window procedure for the frame window. */
/* Called by PM whenever a message is sent to the frame. */
*****/
MRESULT EXPENTRY LocalWndProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
    char        szDir[CCHMAXPATH];
    char        szBuffer[256];
    PDRAGINFO   dragInfo;
    PDRAGITEM   dragItem;
    USHORT      usOp;
    USHORT      usIndicator, cItems, i;
    ULONG       ulBytes;

    switch(msg)
    {
/*****
/* Someone's dragging an object over us. */
*****/
        case DM_DRAGOVER:
            dragInfo = (PDRAGINFO)mp1;

            /* Get access to the DRAGINFO data structure */
            DrgAccessDragInfo(dragInfo);

            /* Can we accept this drop? */
            switch (dragInfo->usOperation)
            {
                /* Return DOR_NODROPOP if current operation */
                /* is link or unknown */
                case DO_UNKNOWN:
                    DrgFreeDraginfo(dragInfo);

```

```

        return (MRFROM2SHORT (DOR_NODROPOP, 0));
        break;

/* Our default operation is Move */
case DO_DEFAULT:
    dragItem = DrgQueryDragitemPtr(dragInfo, 0);
    ulBytes = DrgQueryStrName(dragItem->hstrContainerName,
                              sizeof(szDir),
                              szDir);

    if (!ulBytes)
        return (MRFROM2SHORT (DOR_NODROPOP, 0));
    else
        usOp = DO_MOVE;
        break;

/* Do the requested specific operation */
case DO_MOVE:
case DO_COPY:
    usOp = dragInfo->usOperation;
    break;
}

usIndicator = DOR_DROP;
cItems = DrgQueryDragitemCount(dragInfo);

/* Now, we need to look at each item in turn */
for (i = 0; i < cItems; i++)
{
    dragItem = DrgQueryDragitemPtr(dragInfo, i);

    /* Make sure we can move for a Move request */
    /* or copy for a Copy */
    if (((dragItem->fsSupportedOps & DO_COPYABLE) &&
        (usOp == (USHORT)DO_COPY)) ||
        ((dragItem->fsSupportedOps & DO_MOVEABLE) &&
        (usOp == (USHORT)DO_MOVE)))
    {
        /* Check the rendering format */
        if (DrgVerifyRMF(dragItem, "DRM_OS2FILE", "DRF_UNKNOWN"))
            usIndicator = DOR_DROP;
        else
            usIndicator = DOR_NEVERDROP;
    }
    else
        usIndicator = DOR_NODROPOP;
}

/* Release the draginfo data structure */
DrgFreeDraginfo(dragInfo);

return (MRFROM2SHORT(usIndicator, usOp));
break;

/* Dragged object just left */
case DM_DRAGLEAVE:
    return (MRESULT)FALSE;
    break;

/* Drop the object on us (receive the object) */
case DM_DROP:

/* Get access to the DRAGINFO data structure */
DrgAccessDraginfo(dragInfo);

/* Can we accept this drop? */
switch (dragInfo->usOperation)
{
    /* Return DOR_NODROPOP if current operation */
    /* is link or unknown */
    case DO_UNKNOWN:
        DrgFreeDraginfo(dragInfo);
        return (MRFROM2SHORT (DOR_NODROPOP, 0));
        break;

    /* Our default operation is Move */
    case DO_DEFAULT:

```

```

        dragItem = DrgQueryDragitemPtr(dragInfo, 0);
        ulBytes = DrgQueryStrName(dragItem->hstrContainerName,
                                   sizeof(szDir),
                                   szDir);

        if (!ulBytes)
            return (MRFROM2SHORT (DOR_NODROPOP, 0));
        usOp = (USHORT)DO_MOVE;
        break;

    /* Do the requested specific operation */
    case DO_MOVE:
    case DO_COPY:
        usOp = dragInfo->usOperation;
        break;
}

usIndicator = DOR_DROP;
cItems = DrgQueryDragitemCount(dragInfo);

/* Now, we need to look at each item in turn */
for (i = 0; i < cItems; i++)
{
    dragItem = DrgQueryDragitemPtr(dragInfo, i);

    /* Make sure we can move for a Move request */
    /* or copy for a Copy */
    if (((dragItem->fsSupportedOps & DO_COPYABLE) &&
         (usOp == (USHORT)DO_COPY)) ||
        ((dragItem->fsSupportedOps & DO_MOVEABLE) &&
         (usOp == (USHORT)DO_MOVE)))
    {
        /* Check the rendering format */
        if (DrgVerifyRMF(dragItem, "DRM_OS2FILE", "DRF_UNKNOWN"))
            usIndicator = DOR_DROP;
        else
            usIndicator = DOR_NEVERDROP;
    }
    else
        usIndicator = DOR_NODROPOP;

    /* *****
    /* This is where we would actually move or copy the file,
    /* but we just display the name instead.
    /* *****
    DrgQueryStrName(dragItem->hstrSourceName, 255, szBuffer);
    WinMessageBox(HWND_DESKTOP,
                  HWND_DESKTOP,
                  szBuffer,
                  "Dropped",
                  0,
                  MB_OK);
}

/* Release the draginfo data structure */
DrgFreeDraginfo(dragInfo);

return (MRFROM2SHORT(usIndicator, usOp));
break;

/* Send the message to the usual WC_FRAME WndProc */
default:
    return (*SysWndProc)(hwnd, msg, mp1, mp2);
    break;
}
return (*SysWndProc)(hwnd, msg, mp1, mp2);
}

=====
TARGET.RC
=====
#include <os2.h>
#include "target.h"

ICON TRASHCAN    trashcan.ico

=====
TARGET.H

```

```

=====
#define TRASHCAN 100

=====
TARGET.DEF
=====
NAME      TARGET    WINDOWAPI

DESCRIPTION 'PM Drag and Drop Sample'

CODE      MOVEABLE
DATA      MOVEABLE MULTIPLE

STACKSIZE 24576
HEAPSIZE  10240

PROTMODE

=====
TARGET.LNK
=====
target.obj
target.exe
target.map
target.def

```

Drawing in Windows

This chapter describes, at a high level, the functions specifically intended for drawing in PM windows. For information on the complete set of drawing functions, see the *Graphics Programming Interface Programming Guide*.

About Window-Drawing Functions

The functionality of the PM window-drawing functions overlaps that of similar Graphic Programming Interface (GPI) drawing functions in OS/2. These window-drawing functions are less general than the GPI functions and are somewhat easier to use, but they also offer fewer capabilities than the complete set of GPI functions. Programmers requiring optimum functionality should use the GPI functions.

Points

All drawing in a window takes place in the context of the window's coordinate system. Locations of points in the window are described by POINTL structures, which contain an x and a y coordinate for the point. The lower-left corner of a window always has the coordinates (0,0).

The WinMapWindowPoints function converts the coordinates of points from one window-coordinate space to those of another window-coordinate space. If one of the specified windows is HWND_DESKTOP, the function uses screen coordinates. This function is useful for converting window coordinates to screen coordinates or the other way around.

Rectangles

Locations of window rectangles are described by RECTL structures, which contain the coordinates of two points that define the lower-left and upper-right corners of the rectangle. An empty rectangle is one that has no area: either its right coordinate is less than or equal to its left coordinate, or its top coordinate is less than or equal to its bottom coordinate.

There are two types of rectangles in OS/2: inclusive-inclusive and inclusive-exclusive. In inclusive-exclusive rectangles, the lower-left coordinate of the rectangle is included within the rectangle area, while the upper-right coordinate is excluded from the rectangle area. In an inclusive-inclusive rectangle, both the lower-left and upper-right coordinates are included in the rectangle.

In general, graphics operations involving device coordinates (such as regions, bit maps and bit blts, and window management) use inclusive-exclusive rectangles. All other graphics operations, such as GPI functions that define paths, use inclusive-inclusive rectangles.

Using Window-Drawing Functions

This section explains how to use drawing functions to fill (paint) a rectangle with color, scroll the contents of a window, draw bit maps and text, and determine the dimensions of a rectangle.

Working with Points and Rectangles

The operating system includes functions for manipulating rectangles, many of which change the rectangle coordinates. Other functions draw in a presentation space, using a rectangle to position the drawing operation.

The rest of the rectangle functions are mathematical and do not draw. They are used to manipulate and combine rectangles to produce new rectangles that you then can use in drawing operations.

Determining the Dimensions of a Rectangle

You can calculate the dimensions of an inclusive-exclusive rectangle as follows:

```
cx = rcl.xRight - rcl.xLeft; /* width */
cy = rcl.yTop - rcl.yBottom; /* height */
```

You can calculate the dimensions of an inclusive-inclusive rectangle as follows:

```
cx = (rcl.xRight - rcl.xLeft) + 1; /* width */
cy = (rcl.yTop - rcl.yBottom) + 1; /* height */
```

Filling a Rectangle

The WinFillRect function fills (paints) a rectangle with a specified color. For example, to fill an entire window with blue in response to a WM_PAINT message, you could use the following code fragment, which is taken from a window procedure:

```

HPS    hps;
RECTL  rcl;

case WM_PAINT:
    hps = WinBeginPaint(hwnd, (HPS) NULL, (PRECTL) NULL);
    WinQueryWindowRect(hwnd, &rcl);
    WinFillRect(hps, &rcl, CLR_BLUE);
    WinEndPaint(hps);
    return 0;

```

A more efficient way of painting a client window is to pass a rectangle to the WinBeginPaint function. The rectangle is set to the coordinates of the rectangle that encloses the update region of the window. Drawing in this rectangle updates the window, which can make drawing faster if only a small portion of the window needs to be painted. This method is shown in the following code fragment. Notice that WinFillRect uses the presentation space and a rectangle defined in window coordinates to guide the paint operation.

```

HPS    hps;
RECTL  rcl;

case WM_PAINT:
    hps = WinBeginPaint(hwnd, (HPS) NULL, &rcl);
    WinFillRect(hps, &rcl, CLR_BLUE);
    WinEndPaint(hps);
    return 0;

```

You could draw the entire window during the WM_PAINT message, but the graphics output would be clipped to the update region.

The default method of indicating that a particular portion of a window has been selected is using the WinInvertRect function to invert the rectangle's bits.

Scrolling the Contents of a Window

An application typically responds to a click in a scroll bar by scrolling the contents of the window. This operation has three parts. First, the application changes its internal data-representation state to show what portion of the image must now be in the window. Next, the application moves the current image in the window. Finally, the application draws in the area that has been uncovered by the scrolling operation.

For example, a simple text editor might display a small portion of several pages of text in a window. When the user clicks the Down arrow of the vertical scroll bar, the application moves all the text up one line and displays the next line at the bottom of the window.

This clicking also causes a message to be sent to the client window of the frame window that owns the scroll bar. The application responds to this message by changing its internal data-representation state to show which line of text is topmost in the window, scrolling the text in the window up one line, and drawing the new line at the bottom of the window. There normally is no need to completely redraw the entire window, because the scrolled portion of the image remains valid.

You can use the WinScrollWindow function to scroll the contents of your application windows. WinScrollWindow scrolls a specified rectangular area of the window by a specified x and y offset (in window coordinates). If you set the SW_INVALIDATERGN flag for this function, the areas you uncover by scrolling are added to the window's update region automatically, causing a WM_PAINT message for the areas to be sent to the window.

For example, as used in the simple text editor described previously, the following call scrolls the text up one line (assuming that the *iVScrollInc* parameter specifies the height of the current font) and adds the uncovered area at the bottom of the window to the update region.

```

HWND  hwnd;
LONG  iVScrollInc;

/* Scroll, adding a new area to the update region. */

WinScrollWindow(hwnd, /* Window handle */
0, /* x displacement */

```



```

-(iVScrollInc),      /* y displacement */
(PRECTL) NULL,      /* Scroll rectangle is entire window */
(PRECTL) NULL,      /* Clip rectangle is entire window */
(HRGN) NULL,        /* Update region */
(PRECTL) NULL,      /* Update rectangle */
SW_INVALIDATERGN); /* Scroll-window flag */

```

When the uncovered area is added to the window's update region, a WM_PAINT message is sent to the window. Upon receiving the message, the window draws the line of text at the bottom of the window. If the window has the WS_SYNCPAINT style, the WM_PAINT message is sent to the window before WinScrollWindow returns.

To optimize scrolling speed for repeated scrolling operations, you can omit the SW_INVALIDATERGN flag from the call to WinScrollWindow, which prevents the function from adding the invalid region (uncovered by the scroll) to the window's update region. If you omit the SW_INVALIDATERGN flag, you must pass a region or rectangle to WinScrollWindow. The rectangle or region will contain the area that must be updated after scrolling.

Drawing a Bit Map

The WinDrawBitmap function draws a bit map, identified by a bit map handle, in a specified rectangle. This function enables you to reduce or enlarge the bit map from the source rectangle to the destination rectangle. WinDrawBitmap also can draw in several different copy modes, including using the OR operator to combine source and destination pels.

Drawing Text

There are many ways to draw text in a window in an OS/2 application. The simplest way is to use the WinDrawText function, which draws a single line of text in a specified rectangle, using a variety of alignment methods.

WinDrawText allows you to set a flag so that the function does not draw any text; instead, the function returns the number of characters in the string that will fit in the specified rectangle. For a section of running text, an application can alternate between computation and calls to WinDrawText to draw successive lines of text. When performing this kind of repetitive operation, you can set the DT_WORDBREAK flag in the WinDrawText function to put line breaks on word boundaries rather than between arbitrary characters.

Dynamic Data Exchange

The Dynamic Data Exchange (DDE) protocol uses messages to communicate between applications that share data and uses shared memory as the means of exchanging data between applications. Applications can use DDE for one-time data transfers and for ongoing exchanges in which the applications send updates to each other as new data becomes available. This chapter explains how to use DDE in PM applications.

About Dynamic Data Exchange

DDE is different from the clipboard data-transfer component that is also part of this operating system. The clipboard is almost always used as a one-time response to a specific action by the user, such as choosing **Paste** from a menu. DDE, on the other hand, is often initiated by a user but typically continues without the user's further involvement. DDE is separate from and does not use the clipboard.

DDE always takes place between two applications: a *client* application and a *server* application. The client initiates the exchange by requesting that the server perform a particular action, such as supply data. The client's request to the server is called a *transaction*. If it is able, the server responds by performing the requested action. The important distinction between a client and a server is that the client always initiates DDE transactions.

A server can have many clients simultaneously, and a client can request data from multiple servers. An application can be both a client and a server at the same time. For example, one application could receive data from another application as a client and then act as a server by passing the data to yet another application.

Client and Server Interaction

A DDE conversation actually takes place between two windows: one for each of the participating applications. Applications open a window for each conversation in which they engage. Because a window is identified by its handle, these windows are not necessarily visible. The window belonging to the server application is the *server window*. The window belonging to the client application is the *client window*.

DDE System Example

DDE has many potential uses in real-time data acquisition applications. Consider the example of a DDE-based, real-time system for tracking portfolios. Two hypothetical PM applications cooperate in this example. One application, named the *collector*, is a specialized interface that draws data from an online data service. The other application is a spreadsheet. Both applications use the DDE protocol. The following figure shows the sample spreadsheet layout:

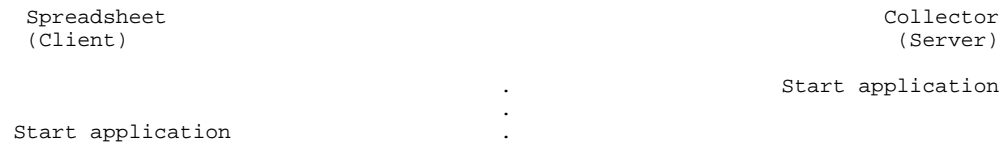
	A	B	C	D
1	<u>Stock</u>	<u>Shares</u>	<u>Price</u>	<u>Extension</u>
2	ABCD	1000	148	148000
3	EFGH	2000	26	52000
4	IJKL	200	24	4800
5	MNOP	2000	93	186000
6				390800

Without DDE, this spreadsheet could be updated by using the clipboard to manually copy numbers from the screen display of the collector application to the spreadsheet. This would require screen sharing or switching between applications. The user also would have to pay close attention to the price data, then undertake the data exchange personally whenever the price data changes.

With DDE, this system could be much more automatic, providing the spreadsheet with the current values for multiple data items, without user intervention. DDE enables the user to set up an exchange between the two applications that updates the spreadsheet whenever a change occurs in the value of specified stocks. After this connection is established, the cell values in the spreadsheet always reflect the most current data available from the collector. This system facilitates the timely analysis of real-time data.

The usefulness of the DDE protocol is not restricted to specialized real-time data-acquisition applications. Productivity software, in general, can benefit significantly from the protocol. For example, a monthly report is prepared using word processor, and the report includes graphs generated in a separate business-graphics package. Without DDE, someone must manually copy and paste each month's new graphs into each month's report. With DDE, the word processor can establish a permanent link to the graphics application so that any changes made to the graphs are reflected in the word-processing document, either automatically or on request.

The following diagram shows a detailed view of the workings of the DDE protocol and describes the collector and spreadsheet interaction and illustrates the forwarding of stock quotes from the collector application to the spreadsheet. For simplicity, this example is limited to the exchange of quotes for a single stock, ABCD.



Load stock-portfolio document	.	
	.	
Calls WinDdeInitiate which	.	
sends WM_DDE_INITIATE	.	
	.	
	.	Accepts and calls
	.	WinDdeRespond
	.	with positive
	.	WM_DDE_INITIATEACK
	.	
Calls WinDdePostMsg which	.	
posts WM_ADVICE	.	
	.	
Request to send info each time	.	
data item ABCD changes; send	.	
in format DDEFMT_TEXT	.	
	.	
	.	Records request in
	.	database and calls
	.	WinDdePostMsg which posts
	.	WM_DDE_ACK
	.	
	.	Whenever data changes
	.	for item ABCD, calls
	.	WinDdePostMsg which posts
	.	WM_DDE_DATA
	.	
Retrieves information from	.	
shared-memory object indicated	.	
by pointer in WM_DDE_DATA	.	
	.	
Updates value of ABCD in	.	
spreadsheet	.	
	.	
When ready to end updates,	.	
calls WinDdePostMsg which posts	.	
WM_DDE_UNADVISE	.	
	.	
	.	Removes request from
	.	database and calls
	.	WinDdePostMsg which posts
	.	WM_DDE_ACK
	.	
To end DDE transaction,	.	
calls WinDdePostMsg which posts	.	
WM_DDE_TERMINATE	.	
	.	
	.	Responds by calling
	.	WinDdePostMsg which posts
	.	WM_DDE_TERMINATE

The collector DDE server application is started first. Typically, applications designed to operate as dedicated DDE servers have a user interface for initialization, and then run minimized. As part of the initialization process, the collector DDE server application performs the necessary tasks (such as entering passwords and testing) to ensure that it can provide data to clients.

The spreadsheet is started next, and the stock-portfolio document is loaded. At this time, the spreadsheet calls WinDdeInitiate, which sends a WM_DDE_INITIATE message to all top-level frame windows, that is, frame windows that have HWND_DESKTOP as their parent. The WM_DDE_INITIATE message is a request to initiate an exchange with an application on a specified topic-in this case, STOCKS. An application can accept this message by responding with a positive WM_DDE_INITIATEACK message, or decline the message by passing the message on to WinDefWindowProc. If no application accepts the request, the spreadsheet assigns an error value to the external reference and its DDE activity concludes.

If the collector acknowledges the request, the spreadsheet can use the newly established exchange to request that the collector provide continuous updates on a specified data item. To make this request, the spreadsheet posts a WM_DDE_ADVISE message to the collector (actually, to a window within the collector that acts as the message recipient for DDE messages), indicating that updates must be sent every time there is a new value available for the data item, ABCD, and that the updates should be in a particular format-for example, DDEFMT_TEXT.

Upon receiving this message, the collector application records the request in its database and posts a WM_DDE_ACK message to the

spreadsheet. From then on, whenever the collector receives a new ABCD stock quote, it posts a WM_DDE_DATA message to the window in the spreadsheet that initiated the exchange. Each of these messages carries a pointer to a shared-memory object that contains the data, rendered in the requested format. When the spreadsheet receives such a message, it retrieves the data from the referenced memory object and uses the data to update the value of the cell containing the external reference.

The periodic updates continue until the spreadsheet document is closed. At that point, the spreadsheet application posts a WM_DDE_UNADVISE message to the collector application, indicating that further updating is unnecessary. Upon receipt of this message, the collector application removes the corresponding data request from its database and posts a positive WM_DDE_ACK message back to the spreadsheet.

Finally, unless the spreadsheet initiates other data exchanges under this same topic, it posts a WM_DDE_TERMINATE message to the collector application, indicating the end of the DDE transaction. The collector application responds with a WM_DDE_TERMINATE message.

Note: At any time during the transaction, both the spreadsheet and collector are free to post a WM_DDE_TERMINATE message to the other application.

Applications, Topics, and Items

DDE uses the three-level hierarchy-*application*, *topic*, and *item*-to uniquely identify a unit of data. An *application* is the name of the server from which the data is desired. A *topic* is a logical data context. For applications that operate on file-based documents, topics are usually file names; for other applications, they are other application-specific strings. An *item* is a data object that can be passed in a DDE transaction. For example, an item might be a single integer, a string, several paragraphs of text, or a bit map. In the collector and spreadsheet model described in the previous section, the application name is *collector*, the topic name is *STOCKS*, and the item name is *ABCD*.

The System Topic

The *system topic* provides a context for information that might be of general interest to any partners in a DDE transaction. Server applications are encouraged to support the system topic at all times. The string used for the system topic is defined in the PM header files as SZDDSYS_TOPIC.

DDE applications should request an exchange on the system topic with a zero-length application name when they start up, to find out what kinds of information other DDE-capable programs can provide.

The system topic must support the items in the following table as well as any other items the application uses:

Item	Description
SZDDSYS_ITEM_FORMATS	A list of strings equivalent to CF_CONSTANTS with the CF_ prefix removed. For example, CF_TEXT = TEXT.
SZDDSYS_ITEM_HELP	A text description of the server's DDE services.
SZDDSYS_ITEM_PROTOCOLS	A list of protocol names the server supports. A protocol is a set of DDE execute commands, each having a standard meaning.
SZDDSYS_ITEM_RESTART	A string that a client can pass to DosExecPgm to invoke a server that is not running.
SZDDSYS_ITEM_RTMSG	Supporting detail for the most recently issued WM_DDE_ACK message. This is useful when more than 8 bits of application-specific return code are required.
SZDDSYS_ITEM_SECURITY	A security-sensitive server application. Any client can initiate a conversation with a

	security-sensitive server, but the server responds only to the Security topic. Typically, the server requires a password from the client before any further data exchange can take place.
SZDDSYS_ITEM_STATUS	An indication of the current status of the server: "Ready" or "Busy".
SZDDSYS_ITEM_SYSITEMS	A list of the items supported under the system topic by this server.
SZDDSYS_ITEM_TOPICS	A list of the topics currently supported by the application. This can vary from moment to moment.

Individual elements of lists should be delimited by tabs, as in the DDEFMT_TEXT format.

DDE Initiation

A client application initiates a DDE conversation by calling WinDdeInitiate, specifying the server application-name string and the topic-name string. WinDdeInitiate fills a DDEINIT data structure with the specified strings, then sends a WM_DDE_INITIATE message to all frame windows that have HWND_DESKTOP as their parent. The message contains the handle of the client application and a pointer to the DDEINIT data structure. The following figure illustrates the DDEINIT data structure:

```
typedef struct _DDEINIT
{
    ULONG    cb;
    PSZ      pszAppName;
    PSZ      pszTopic;
    USHORT   usConvContext;
} DDEINIT;
```

Because the message is sent rather than posted, WinDdeInitiate requires a response from all recipients of the message before it returns control to the client application.

Any potential server must contain a *server window*, a top-level frame window that has been subclassed to receive and process WM_DDE_INITIATE messages. When a server window receives WM_DDE_INITIATE, it examines the application-name and topic-name strings in the DDEINIT data structure. If the application-name string matches and the server supports the requested topic, the server acknowledges the client's request.

Either the application-name or topic-name string can be zero-length. If the application-name string is zero-length, all servers check the topic-name string. Each server that supports the topic sends a separate acknowledgment to the client. If the topic-name string is zero-length, the server sends an acknowledgment for each supported topic. Using zero-length strings, a client can obtain the names of all the active servers in the system or the names of all the topics a server supports.

The following pseudocode shows how servers respond to WM_DDE_INITIATE messages:

```
If (specific app requested and server is instance of app) or
    (specific app not requested)
{
    If (specific topic requested)
        If (server can support topic)
            acknowledge the requested topic
    else
        acknowledge each supported topic
}
```

A server acknowledges its support of a specific topic by calling WinDdeRespond, specifying the handle of its server window, its application name, and the name of the supported topic. WinDdeRespond fills a DDEINIT data structure with the specified strings, then sends a WM_DDE_INITIATEACK message to the client. The message contains the handle of the server window and a pointer to the DDEINIT data structure. The client examines the topic-name string in the DDEINIT data structure and decides whether to begin a transaction on the topic.

If two applications agree on some unspecified protocol and can exchange window handles by some means, they can use DDE messages on those window handles without going through an initiate sequence.

An application does not need to fill in a DDEINIT data structure; WinDdeInitiate and WinDdeRespond automatically fill the data structure. However, applications must extract the application name and topic name from the DDEINIT data structure when receiving a WM_DDE_INITIATE or WM_DDE_INITIATEACK message.

Shared-Memory Object

After initiating a conversation, the client interacts with the server by issuing transactions. A *transaction* is a client's request that the server perform a particular action.

To issue a transaction, the client allocates a shared-memory object, writes data about its request to the object using a DDESTRUCT data structure, then calls WinDdePostMsg to post a transaction message to the server. The transaction message contains the client-window handle and a pointer to the shared-memory object. When the server receives the message, it uses the pointer to access the shared-memory object.

The server responds by allocating a shared-memory object, writing its response to the object using a DDESTRUCT data structure, then calling WinDdePostMsg to post a response message to the client. The response message contains the server-window handle and a pointer to the shared-memory object.

A DDESTRUCT data structure occupies the first part of the memory object. Next comes the item-name string, followed by the actual data being exchanged. The offset fields of the DDESTRUCT data structure must be set to point to the name string and the beginning of the data. The *cbData* field also must be set to indicate the number of bytes of data.

The sender of a DDE transaction message must allocate a shared-memory object using DosAllocSharedMem, then call DosGiveSharedMem to share the object with the receiving application. To share an object, the sender must know the process identifier of the recipient. The process identifier can be obtained by calling WinQueryWindowProcess for the recipient's window handle. WinDdePostMsg also gives the memory object.

The sender should not try to access the object after sending it to the recipient in a DDE message. After posting a transaction message, WinDdePostMsg automatically frees the shared-memory object from the sender's virtual address space. An application need not call DosFreeMem for this purpose. However, the recipient must call DosFreeMem when it is finished using the object.

Transaction Status Flags

DDE client and server applications can specify status flags in the DDESTRUCT data structure. These flags are constant values that applications use to control various aspects of a DDE transaction. They can be combined in the *fsStatus* word of the DDESTRUCT data structure by using the OR operator. The following table lists the DDE status flags:

Flag Name	Description
DDE_FACK	Indicates a positive acknowledgment.
DDE_FACKREQ	Requests an acknowledgment from the receiving application.
DDE_FAPPSTATUS	Indicates that the upper 8 bits of the status word are used for application-specific data.
DDE_FBUSY	Indicates that the application received a request but cannot respond because it is busy filling an earlier request.
DDE_FNODATA	Indicates that no data is to be transferred in response to the WM_DDE_ADVISE message.

DDE_FRESERVED	Reserved; must be 0.
DDE_FRESPONSE	Indicates a response to a WM_DDE_REQUEST message.
DDE_NOTPROCESSED	Indicates that the message received is not supported.

Transaction and Response Messages

DDE applications use WinDdePostMsg to communicate during data-exchange transactions. A client application posts transaction messages to a server, which responds by posting acknowledgment messages to the client. Transaction and acknowledgment messages have the same data structure. The first message parameter contains the handle of the sending window; the second contains a pointer to the shared-memory object that contains message information.

The DDE protocol defines five transaction types:

- Advise
- Unadvise
- Request
- Poke
- Execute

These transactions are permitted only within an exchange begun by using the WM_DDE_INITIATE message. Each transaction type has a corresponding message that a client uses to initiate the transaction with a server:

- WM_DDE_ADVISE
- WM_DDE_UNADVISE
- WM_DDE_REQUEST
- WM_DDE_POKE
- WM_DDE_EXECUTE

A server acknowledges a transaction message by posting a WM_DDE_ACK message to the client. The client must examine the status field of the DDESTRUCT data structure to determine whether the response is positive or negative.

A server application posts a WM_DDE_DATA message to the client to indicate that requested data is available. If the status bit of the DDESTRUCT structure has the DDE_FACKREQ flag set, the client must acknowledge receipt of the data by sending a WM_DDE_ACK message to the server.

The fifth parameter of WinDdePostMsg is a flag used to specify whether to try to post a message again if the first attempt failed because the destination queue was full (server returns the DDE_FBUSY flag). If the retry flag is set, WinDdePostMsg posts the message at 1-second intervals until the message is posted successfully.

The following sections explain the five basic types of DDE transactions and the messages involved with each. These messages are posted with WinDdePostMsg, which automatically builds and fills a DDEINIT data structure.

Request and Poke Transactions

A client application can use the DDE protocol to obtain a data item from a server (WM_DDE_REQUEST) or to submit a data item to a server (WM_DDE_POKE).

The client posts a WM_DDE_REQUEST message to the server, specifying an item and format by allocating a shared-memory object, filling in a DDESTRUCT data structure, and passing the data structure to WinDdePostMsg.

If the server is unable to satisfy the request, it sends the client a negative WM_DDE_ACK message. If the server can satisfy the request, it renders the item in the requested format, includes it with a DDESTRUCT data structure in a shared-memory object, and posts a WM_DDE_DATA message to the client.

Upon receiving a WM_DDE_DATA message, the client processes the data item. At the beginning of the shared-memory object, the

DDESTRUCT data structure contains a status word indicating whether the sender requested an acknowledgment message. If the DDE_FACKREQ bit of the status word is set, the client must send the server a positive WM_DDE_ACK message.

Upon receiving a negative WM_DDE_ACK message, the client can ask for the same item again, specifying a different DDE format. Typically, a client first asks for the most complex format it can support, then steps down, if necessary, through progressively simpler formats, until it finds one the server can provide.

Advise and Unadvise Transactions

A client application can use DDE to establish a link to an item in a server application. When such a link is established, the server sends periodic updates about the linked item to the client (typically, whenever the data associated with the item in the server application has changed). A permanent *data stream* is established between the two applications and remains in place until it is explicitly disconnected.

The client sends the server a WM_DDE_ADVISE message to set up the data link. The advise message contains a shared-memory pointer containing a DDESTRUCT data structure with the item name, format information, and status information.

If the server has access to the requested item and can render it in the desired format, the server records the new link, then sends the client a positive WM_DDE_ACK message. Until the client issues a WM_DDE_UNADVISE message, the server sends data messages to the client every time a change occurs in the source data associated with the item in the server application.

If the server is unable to satisfy the request, it sends the client a negative WM_DDE_ACK message.

When a link is established with the DDE_FNODATA status bit cleared, the client is sent the data each time the data changes. In such cases, the server renders the new version of the item in the previously specified format and posts a WM_DDE_DATA message to the client.

When the client receives a WM_DDE_DATA message, it extracts data from the shared-memory object by using the DDESTRUCT data structure at the beginning of the object. If the DDE_FACKREQ status bit in the status word of the DDESTRUCT data structure is set, the client must post a positive WM_DDE_ACK message to the server.

When a link is established with the DDE_FNODATA status flag set, a notification, not the data itself, is posted to the client each time the data changes. In this case, the server does not render the new version of the item when the source data changes, but simply posts a WM_DDE_DATA message with 0 bytes of data and the DDE_FNODATA status flag set.

The client can request the latest version of the data by performing a regular one-time WM_DDE_REQUEST transaction, or it can simply ignore the data-change notice from the server. In either case, if the DDE_FACKREQ status bit is set, the client should send a positive WM_DDE_ACK message to the server.

When a client sends a WM_DDE_ADVISE message on a topic/item pair that is already engaged in an advise loop but has a different format specified, the server interprets this as a request to add an advise loop with the given format requested. Therefore, several advise loops can exist for a given topic/item pair. If a server does not support this extent of advise loops, it rejects the advise request.

Correspondingly, when a server receives a WM_DDE_UNADVISE message, the server must compare the format field with the current format of the advise loop. Only if the specified format is 0, meaning all advise loops, or matches an active advise loop does the server stop the advise loop and return a positive acknowledgment.

To terminate a specific item link, the client posts a WM_DDE_UNADVISE message to the server. The server ensures that the client currently has a link to the specified item in this exchange. If the link exists, the server sends a positive WM_DDE_ACK message to the client and no longer sends updates on the item in this exchange. If the server has no such link, it sends a negative WM_DDE_ACK message.

To terminate all links for a particular exchange, the client application posts a WM_DDE_UNADVISE message with a zero-length item name to the server. The server ensures that the exchange has at least one link currently established. If so, the server posts a positive WM_DDE_ACK message to the client, and no longer sends any updates in the exchange. If the server has no links in the exchange, it posts a negative WM_DDE_ACK message.

Execute Transaction

A PM application can use the DDE protocol to cause commands to be executed in another application. Such remote executions are performed by the WM_DDE_EXECUTE transaction.

To execute a remote command, the client application posts to the server a WM_DDE_EXECUTE message containing a pointer to a shared-memory object that contains a DDESTRUCT data structure and a command string.

The server attempts to execute the specified string according to some agreed-upon protocol. If successful, the server posts a positive WM_DDE_ACK message to the client. If unsuccessful, a negative WM_DDE_ACK message is posted.

DDE Termination

At any time, either the client or the server may terminate an exchange by issuing a WM_DDE_TERMINATE message. Similarly, both the client application and server application must be able to receive a WM_DDE_TERMINATE message at any time.

An application must end its exchanges before terminating. The application posts a WM_DDE_TERMINATE message with a zero-length shared-memory pointer. A WM_DDE_TERMINATE message stops all transactions for a given exchange.

The WM_DDE_TERMINATE message means that the sender sends no further messages in that exchange and that the recipient can destroy its DDE window. The recipient must always send a WM_DDE_TERMINATE message promptly in response; it is not permissible to send a negative, busy, or positive WM_DDE_ACK message instead.

If the original sender of the termination request receives any other message before the WM_DDE_TERMINATE message arrives from the recipient of the request, it should not respond, because the sender of the other message might have already destroyed the window to which the response would be sent.

Unique Data Formats

Whenever an application exchanges data using the DDE protocol, it must specify the format of the data in the *usFormat* field of the DDESTRUCT data structure. The system-defined standard format for exchanging text data is DDEFMT_TEXT. Applications can also use constant names to specify the format of data to be exchanged listed in the following table:

Data Format Name	Description
SZFMT_BITMAP	Specifies that the data is a bit map.
SZFMT_CPTTEXT	Specifies text whose format is defined by a CPTTEXT data structure. Applications can use this format to pass multiple-language strings without changing the conversation context.
SZFMT_DIF	Specifies that the data is in Data Image Format (DIF).
SZFMT_DSPBITMAP	Specifies that the data is a bit-map representation of a private data format.
SZFMT_DSPMETAFILE	Specifies that the data is a metafile representation of a private data format.
SZFMT_DSPMETAFILEPICT	Specifies that the data is a metafile picture representation of a private data format.
SZFMT_DSPTEXT	Specifies that the data is a text representation of a private data format.
SZFMT_LINK	Specifies that the data is in link-file format.
SZFMT_METAFILE	Specifies that the data is a metafile.
SZFMT_METAFILEPICT	Specifies that the data is a metafile picture defined by an MFP data structure.
SZFMT_OEMTEXT	Specifies that the data is in OEM Text

	format.
SZFMT_PALETTE	Specifies that the data is in palette format.
SZFMT_SYLK	Specifies that the data is in Synchronous Link format.
SZFMT_TEXT	Specifies that the data is an array of text characters. These characters can include new-line characters to indicate linebreaks. The zero-length character indicates the end of the text data.
SZFMT_TIFF	Specifies that the data is in Tag Image File Format (TIFF).

Applications can define their own data formats. However, each nonstandard DDE format must have a unique identification number. To receive an identification number for a nonstandard format, the application must register the name of the format in the system atom table. Other applications that have the name of the format can then query the system atom table for the format's identification number. This method ensures that all applications use the same atom to identify a format.

Synchronization Rules

A window processing DDE requests from another window must process them strictly in the order in which the requests were received.

A window does not need to apply this first-in first-out (FIFO) rule between requests from different windows-that is, it may provide asynchronous support for multiple processes. For example, a window might have the following requests in its queue:

1. Request message from window x
2. Request message from window y
3. Request message from window x

The window must process request message 1 before request message 3, but it does not have to process request message 2 before request message 3. If y has a lower priority than x, the window follows the order 1, 3, 2.

If a server is unable to process an incoming request because it is waiting for an external process, it must post a busy WM_DDE_ACK message to the client, to prevent deadlock. A busy WM_DDE_ACK message can also be sent if the server is unable to process an incoming request quickly.

Language-Sensitive DDE Applications

DDE applications written for the international market must be able to exchange data in several different languages. The CONVCONTEXT data structure, along with WinDdeInitiate and WinDdeRespond, provide this support.

A language-sensitive DDE application defines the context of a conversation by filling a CONVCONTEXT data structure with the appropriate country code and code-page identifiers. The CONVCONTEXT data structure also contains a context flag. If this flag is set to DDECTXT_CASESENSITIVE, applications must compare strings in a case-sensitive manner. Language-sensitive DDE applications use WinDdeInitiate and WinDdeRespond to establish a DDE conversation. These functions pass a pointer to a CONVCONTEXT data structure.

Using Dynamic Data Exchange

This section explains how to perform the following tasks:

- Initiate a DDE conversation

- Create a shared-memory object for DDE
- Send positive acknowledgment messages
- Send negative acknowledgment messages
- Perform a one-time data transfer
- Establish a permanent data link
- Execute commands in a remote application
- Terminate a DDE conversation

Note: Most of the sample code in this section is part of complete programs for either a client application or a server application. Both programs are illustrated in "Sample Code for Dynamic Data Exchange".

Initiating a DDE Conversation

The client application initiates a DDE conversation by calling `WinDdeInitiate`, specifying the server application-name string and the topic-name string.

The sample client application in "Sample Code for Dynamic Data Exchange" allows the user to initiate a DDE conversation from a context menu. The following code fragment shows how the client application processes that request:

```
/* User starts DDE conversation */
case IDM_POLL:
    WinPostMsg(hListWnd, LM_DELETEALL, 0, 0);
    ShowMessage("Polling...");
    context.cb = sizeof(CONVCONTEXT);
    context.fsContext = 0;
    WinDdeInitiate(hwnd, szApp, szTopic, &context);
    ShowMessage("Polling complete.");
    break;
```

The following sample code shows how the server application determines whether to send a positive or negative acknowledgment to the `WinDdeInitiate` call:

```
/* ***** */
/* Check incoming poll - if the App and Topic match, */
/* we must acknowledge. If both are zero-length, the client is */
/* searching for anyone to talk to - send our names */
/* ***** */
szClientApp = pDDEinit->pszAppName;
szClientTopic = pDDEinit->pszTopic;
ShowMessage(szClientApp);
ShowMessage(szClientTopic);

if (!strcmpi(szClientApp, szApp) ||
    !strcmpi(szClientApp, NULL))
{
    if (!strcmpi(szClientTopic, szTopic) ||
        !strcmpi(szClientTopic, NULL))
    {
        context.cb = sizeof(CONVCONTEXT);
        context.fsContext = 0;
        WinDdeRespond(hClientWnd,
                      hwnd,
                      szApp,
                      szTopic,
                      &context);
    }
}
break;
```

Creating a Shared-Memory Object for DDE

The following code fragment shows how to create a shared-memory object for a DDE transaction. The parameters include the destination window for the DDE message, item name for the transaction, status word, format of the data, actual data to be transferred (if any), and the length of the data. The allocated object must be big enough to hold the DDESTRUCT data structure, item name, and the actual data to be transferred. The sample returns a pointer (PDDESTRUCT) to a shared-memory object that is ready to post as part of a DDE message.

```
/* Get some sharable memory */
DosAllocSharedMem((PVOID)&mem,
    NULL,
    sizeof(DDESTRUCT)+21,
    PAG_COMMIT |
    PAG_READ |
    PAG_WRITE |
    OBJ_GIVEABLE);

/* Get the server's ID and give it access to the */
/* shared memory */
WinQueryWindowProcess(hServerWnd, &pid, &tid);
DosGiveSharedMem(&mem, pid, PAG_READ | PAG_WRITE);

/* Setup DDE data structures */
/* (11 byte name length, 10 plus NULL, 10 byte data length) */
pDDEdata = (PDDESTRUCT)mem;
pDDEdata->cbData = 10; /* Data length */
pDDEdata->fsStatus = 0; /* Status */
pDDEdata->usFormat = DDEFMT_TEXT; /* Text format */

/* Go past end of data structure for the name */
pDDEdata->offszItemName = sizeof(DDESTRUCT);

/* Go past end of structure (plus past the name) */
/* for the data */
pDDEdata->offfabData = sizeof(DDESTRUCT)+11;
strcpy((BYTE *) (pDDEdata+pDDEdata->offszItemName)),
    "STATUS");

/* Post our request to the server program */
WinDdePostMsg(hServerWnd,
    hwnd,
    WM_DDE_REQUEST,
    pDDEdata,
    DDEPM_RETRY);
```

Sending a Positive Acknowledgment

You can send a positive acknowledgment by posting a WM_DDE_ACK message with the DDE_FACK and DDE_FRESPONSE flags set in the status word of the shared-memory data structure. The following code fragment shows how to do so:

```
/* Specify the status flags, when allocating shared memory */
pDDEdata->fsstatus = DDE_FACK | DDE_FRESPONSE;
.
.
.

/* Post the message */
WinDdePostMsg(hwndDest, /* Handle of destination */
    hwndSource, /* Handle of source */
    WM_DDE_ACK, /* Message */
    pDDEdata, /* Shared-memory pointer */
    DDEPM_RETRY); /* Retry */
```

Sending a Negative Acknowledgment

You can send a negative acknowledgment by posting a WM_DDE_ACK message with the DDE_NOTPROCESSED flag set in the status word of the shared-memory data structure. By not specifying DDE_FACK, it is legal to specify DDE_NOTPROCESSED, but only if the message is not supported, such as WM_DDE_POKE for the specified item. DDE_NOTPROCESSED is not the negative respond. The following code fragment shows how to do so:

```
/* Specify the status flag, when allocating shared memory */
pDDEdata->fstatus &= 0;
.
.
.

/* Post the message */
WinDdePostMsg(hwndDest,          /* Handle of destination */
              hwndSource,        /* Handle of source */
              WM_DDE_ACK,        /* Message */
              pDDEdata,          /* Shared-memory pointer */
              DDEPM_RETRY);      /* Retry */
```

If an application is busy when it receives a DDE message, it can post a WM_DDE_ACK message with the DDE_FBUSY flag set.

Performing a One-Time Data Transfer

A client application posts a WM_DDE_REQUEST or WM_DDE_POKE message to perform a one-time data transfer with a server application. The item-name portion of the shared-memory object passed with the message contains the name of the desired item. When the client posts a WM_DDE_POKE message, the data portion of the shared-memory object contains the data being sent to the server.

If the server can satisfy the request, it renders the item in the requested format and includes it, with a DDESTRUCT data structure, in a shared-memory object and posts a WM_DDE_DATA message to the client, as shown in the following code fragment:

```
/* The DDE data structure is passed, and */
/* the client should have shared it with us */
pDDEdata = (PDDESTRUCT)mp2;
szReqItem = (BYTE *) (pDDEdata + (pDDEdata->offsetszItemName));
ShowMessage(szReqItem);

/* We support item status, but not anything else */
if (!strcmpi(szReqItem, szItem))
{
    ShowMessage("sending...");

    /* Get some sharable memory */
    DosAllocSharedMem((PVOID)&mem,
                     NULL,
                     sizeof(DDESTRUCT)+21,
                     PAG_COMMIT |
                     PAG_READ |
                     PAG_WRITE |
                     OBJ_GIVEABLE);

    /* Get the server's id and give it access to the */
    /* shared memory */
    WinQueryWindowProcess(hClientWnd, &pid, &tid);
    DosGiveSharedMem(&mem, pid, PAG_READ | PAG_WRITE);

    /* Setup DDE data structures */
    /* (11 byte name length, 10 plus NULL, 10 byte data length) */
    pDDEdata = (PDDESTRUCT)mem;
    pDDEdata->cbData = 10; /* Data length */
    pDDEdata->fsStatus = 0; /* Status */
    pDDEdata->usFormat = DDEFMT_TEXT; /* Text format */
}
```

```

/* Go past end of structure for the name */
pDDEdata->offszItemName = sizeof(DDESTRUCT);

/* Go past end of structure (and name) for the data */
pDDEdata->offabData = sizeof(DDESTRUCT)+11;
strcpy((BYTE *) (pDDEdata+(pDDEdata->offabData)), szStatus);
WinDdePostMsg(hClientWnd,
              hwnd,
              WM_DDE_DATA,
              pDDEdata,
              DDEPM_RETRY);
}

else
{
    ShowMessage("rejecting...");
    pDDEdata->cbData = 0; /* Data length */
    pDDEdata->fsStatus = DDE_NOTPROCESSED; /* Status */
    pDDEdata->usFormat = DDEFMT_TEXT; /* Text format */
    WinDdePostMsg(hClientWnd,
                  hwnd,
                  WM_DDE_ACK,
                  pDDEdata,
                  DDEPM_RETRY);
}
ShowMessage("sent...");

```

Establishing a Permanent Data Link

The client posts a WM_DDE_ADVISE message to the server to set up a permanent data link. The advise message contains a shared-memory pointer containing a DDESTRUCT data structure with the item name, format information, and status information. The following sample code shows how to establish a link:

```

WinDdePostMsg(hwndServer, /* Handle of server */
              hwndClient, /* Handle of client */
              WM_DDE_ADVISE, /* Message */
              pddeStruct, /* Shared-memory pointer */
              DDEPM_RETRY); /* Retry */

```

When a link is established with the DDE_FNODATA status flag set, a notification, not the data itself, is posted to the client each time the data changes. In this case, the server does not render the new version of the item when the source data changes, but simply posts a WM_DDE_DATA message with 0 bytes of data and the DDE_FNODATA status flag set, as shown in the following code fragment:

```

/* Specify the data length and status flag, */
/* when allocating shared memory */
pDDEdata->cdData = 0;
pDDEdata->fstatus = DDE_FNODATA;
.
.
.

/* Post the message */
WinDdePostMsg(hwndClient, /* Handle of client */
              hwndServer, /* Handle of server */
              WM_DDE_DATA, /* Message */
              pddeStruct, /* Shared-memory pointer */
              DDEPM_RETRY); /* Retry */

```

Terminating a Permanent Link

The client terminates a data link by posting a WM_DDE_UNADVISE message to the server, as shown in the following code fragment:

```
WinDdePostMsg(hwndServer,          /* Handle of server      */
               hwndClient,         /* Handle of client     */
               WM_DDE_UNADVISE,    /* Message              */
               pddeStruct,         /* Shared-memory pointer */
               DDEPM_RETRY);       /* Retry                */
```

Executing Commands in a Remote Application

To execute a remote command, the client application posts to the server a WM_DDE_EXECUTE message containing a pointer to a shared-memory object that contains a DDESTRUCT data structure and a command string, as shown in the following code fragment:

```
WinDdePostMsg(hwndServer,          /* Handle of server      */
               hwndClient,         /* Handle of client     */
               WM_DDE_EXECUTE,     /* Message              */
               pddeStruct,         /* Shared-memory pointer */
               DDEPM_RETRY);       /* Retry                */
```

Terminating a DDE Conversation

At any time, either the client or the server may terminate a DDE conversation by posting a WM_DDE_TERMINATE message, as shown in the following code fragment:

```
WinDdePostMsg(hwndDest,           /* Handle of destination */
               hwndSource,         /* Handle of source      */
               WM_DDE_TERMINATE,   /* Message              */
               NULL,               /* No shared-memory pointer */
               DDEPM_RETRY);       /* Retry                */
```

Sample Code for Dynamic Data Exchange

This section illustrates a complete sample program for both client and server applications involved in dynamic data exchange (DDE). Several parts of this program are explained in "Using Dynamic Data Exchange".

Client Application Sample Code

The client application includes the following files:

- DDEC.C
- DDEC.RC
- DDEC.H
- DDEC.DEF
- DDEC.LNK
- DDEC.MAK

The following sample shows the client application code:

```

=====
DDEC.C
=====

#define INCL_WIN
#define INCL_DOS

#include <os2.h>
#include <stdio.h>
#include "ddec.h"

#pragma linkage (main,optlink)
INT main(VOID);
void ShowMessage(PSZ);

/*****
/* Main() - program entry point. */
*****/
MRESULT EXPENTRY LocalWndProc(HWND, ULONG, MPARAM, MPARAM);

HAB hab;
HWND hFrameWnd, hListWnd, hServerWnd;
PFNWP SysWndProc;

INT main (VOID)
{
    HMQ hmq;
    FRAMECDATA fcd;
    QMSG qmsg;

    if (!(hab = WinInitialize(0)))
        return FALSE;

    if (!(hmq = WinCreateMsgQueue(hab, 0)))
        return FALSE;

    /*****
    /* Setup the frame control data for the frame window. */
    *****/
    fcd.cb = sizeof(FRAMECDATA);
    fcd.flCreateFlags = FCF_TITLEBAR
                      | FCF_SYSMENU
                      | FCF_MENU
                      | FCF_SIZEBORDER
                      | FCF_SHELLPOSITION
                      | FCF_MINMAX
                      | FCF_TASKLIST;

    fcd.hmodResources = NULLHANDLE;

    /*****
    /* Set our resource key (so PM can find menus, icons, etc). */
    *****/
    fcd.idResources = DDEC;

    /*****
    /* Create the frame - it will hold the container control. */
    *****/
    hFrameWnd = WinCreateWindow(HWND_DESKTOP,
                                WC_FRAME,
                                "DDE Client",
                                0, 0, 0, 0,
                                NULLHANDLE,
                                HWND_TOP,
                                DDEC,
                                &fcd,
                                NULL);

```



```

/*****
/* Verify that the frame was created; otherwise, stop. */
*****/
if (!hFrameWnd)
    return FALSE;

/*****
/* Set an icon for the frame window. */
*****/
WinSendMsg(hFrameWnd,
            WM_SETICON,
            (MPARAM)WinQuerySysPointer(HWND_DESKTOP,
                                         SPTR_FOLDER,
                                         FALSE),
            NULL);

/*****
/* Create a list window child. */
*****/
hListWnd = WinCreateWindow(hFrameWnd,
                            WC_LISTBOX,
                            NULL,
                            LS_HORZSCROLL,
                            0, 0, 0, 0,
                            hFrameWnd,
                            HWND_BOTTOM,
                            FID_CLIENT,
                            NULL,
                            NULL);

/*****
/* We must intercept the frame window's messages */
/* (to capture any input from the container control). */
/* We save the return value (the current WndProc), */
/* so we can pass it all the other messages the frame gets. */
*****/
SysWndProc = WinSubclassWindow(hFrameWnd, (PFNWP)LocalWndProc);

WinShowWindow(hFrameWnd, TRUE);

/*****
/* Standard PM message loop - get it, dispatch it. */
*****/
while (WinGetMsg(hab, &qmsg, NULLHANDLE, 0, 0))
{
    WinDispatchMsg(hab, &qmsg);
}

/*****
/* Clean up on the way out. */
*****/
WinDestroyMsgQueue(hmq);
WinTerminate(hab);

return TRUE;
}

/*****
/* LocalWndProc() - window procedure for the frame window. */
/* Called by PM whenever a message is sent to the frame. */
*****/
MRESULT EXPENTRY LocalWndProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
    PSZ          szData;

    /* DDE strings */
    PSZ          szApp = "DDEdemo", szTopic = "System";
    PSZ          szInApp, szInTopic;

    /* System-defined DDE structures */
    CONVCONTEXT context;
    PDDEINIT     pDDEinit;
    PDDESTRUCT   pDDEdata;

    /* Server process and thread IDs */
    PID          pid;
    TID          tid;

    /* Pointer to memory we'll allocate */

```

```

ULONG      mem;

switch(msg)
{
    /* All answers to the WinDDEInitate call arrive here */
    case WM_DDE_INITIATEACK:
        pDDEinit = (PDDEINIT)mp2;
        szInApp = pDDEinit->pszAppName;
        szInTopic = pDDEinit->pszTopic;
        ShowMessage("server answered...");
        hServerWnd = (HWND)mp1;
        break;

    /* All answers to DDE requests arrive here */
    case WM_DDE_DATA:
        ShowMessage("data in");
        pDDEdata = (PDDESTRUCT)mp2;
        DosGetSharedMem(pDDEdata, PAG_READ | PAG_WRITE);
        szData = (BYTE *) (pDDEdata+(pDDEdata->offabData));
        ShowMessage(szData);
        break;

    /* Menu item processing */
    case WM_COMMAND:
        switch (SHORT1FROMMP(mp1))
        {
            /* User starts DDE conversation */
            case IDM_POLL:
                WinPostMsg(hListWnd, LM_DELETEALL, 0, 0);
                ShowMessage("Polling...");
                context.cb = sizeof(CONVCONTEXT);
                context.fsContext = 0;
                WinDdeInitiate(hwnd, szApp, szTopic, &context);
                ShowMessage("Polling complete.");
                break;

            /* User requests data from the server */
            case IDM_DATA:

                /* Get some sharable memory */
                DosAllocSharedMem((PVOID)&mem,
                                NULL,
                                sizeof(DDESTRUCT)+21,
                                PAG_COMMIT |
                                PAG_READ |
                                PAG_WRITE |
                                OBJ_GIVEABLE);

                /* Get the server's ID and give it access */
                /* to the shared memory */
                WinQueryWindowProcess(hServerWnd, &pid, &tid);
                DosGiveSharedMem(&mem, pid, PAG_READ | PAG_WRITE);

                /* Setup DDE data structures */
                /* (11 byte name length, 10 plus NULL, */
                /* 10 byte data length) */
                pDDEdata = (PDDESTRUCT)mem;
                pDDEdata->cbData = 10; /* Data length */
                pDDEdata->fsStatus = 0; /* Status */
                pDDEdata->usFormat = DDEFMT_TEXT; /* Text format */

                /* Go past end of structure for the name */
                pDDEdata->offszItemName = sizeof(DDESTRUCT);

                /* Go past end of data structure */
                /* (plus past the name) for the data */
                pDDEdata->offabData = sizeof(DDESTRUCT)+11;
                strcpy((BYTE *) (pDDEdata+(pDDEdata->offszItemName)),
                    "STATUS");

                /* Post our request to the server program */
                WinDdePostMsg(hServerWnd,
                            hwnd,
                            WM_DDE_REQUEST,
                            pDDEdata,
                            DDEPM_RETRY);

                break;
        }
    }
}

```

```

        /* User terminates the conversation */
        case IDM_CLOSE:
            WinDdePostMsg(hServerWnd,
                          hwnd,
                          WM_DDE_TERMINATE,
                          NULL,
                          DDEPM_RETRY);

            break;

        /* User closes the window */
        case IDM_EXIT:
            WinPostMsg(hwnd, WM_CLOSE, 0, 0);
            break;
    }
    break;

    /* Send the message to the usual WC_FRAME WndProc */
    default:
        return (*SysWndProc)(hwnd, msg, mp1, mp2);
        break;
    }

    return FALSE;
}

/***** ShowMessage(). *****/
/***** ShowMessage(PSZ szText) *****/
void ShowMessage(PSZ szText)
{
    WinPostMsg(hListWnd,
               LM_INSERTITEM,
               MPFROMSHORT(LIT_END),
               szText);
}

=====
DDEC.RC
=====
#include <os2.h>
#include "ddec.h"

MENU    DDEC
BEGIN
    SUBMENU    "Commands",    IDM_MENU
    BEGIN
        MENUITEM    "Initiate",    IDM_POLL
        MENUITEM    "Data",    IDM_DATA
        MENUITEM    "Close",    IDM_CLOSE
        MENUITEM    "Exit",    IDM_EXIT
    END
END

=====
DDEC.H
=====
#define DDEC    100
#define IDM_MENU    101
#define IDM_POLL    102
#define IDM_INITIATE    103
#define IDM_DATA    104
#define IDM_CLOSE    105
#define IDM_EXIT    106

=====
DDEC.DEF
=====
NAME DDEC WINDOWAPI

DESCRIPTION 'PM DDE Client Sample'

CODE    MOVEABLE
DATA    MOVEABLE MULTIPLE

STACKSIZE    24576
HEAPSIZE    10240

PROTMODE

```

```

=====
DDEC.LNK
=====
ddec.obj
ddec.exe
ddec.map

ddec.def

=====
DDEC.MAK
=====

CC      = icc /c /Ge /Gd- /Se /Re /ss /Gm+
LINK    = link386
HEADERS = ddec.h

#-----
#   A list of all of the object files.
#-----
ALL_OBJ1 = ddec.obj

all: ddec.exe

ddec.res: ddec.rc ddec.h

ddec.obj: ddec.c $(HEADERS)

ddec.exe: $(ALL_OBJ1) ddec.def ddec.lnk ddec.res
          $(LINK) @ddec.lnk
          rc -p -x ddec.res ddec.exe

```

Server Application Sample Code

The server application includes the following files:

- DDES.C
- DDES.RC
- DDES.H
- DDES.DEF
- DDES.LNK
- DDES.MAK

The following sample shows the server application code:

```

=====
DDES.C
=====

#define INCL_WIN
#define INCL_WINDDE
#define INCL_DOS

#include <os2.h>
#include <stdio.h>
#include <string.h>
#include "ddes.h"

#pragma linkage (main, optlink)
INT main(VOID);
void ShowMessage(PSZ);

/*****
/* Main() - program entry point.
*****/
*****/
MRESULT EXPENTRY LocalWndProc(HWND, ULONG, MPARAM, MPARAM);

```

```

HAB      hab;
HWND     hFrameWnd, hListWnd, hClientWnd;
PFNWP    SysWndProc;

INT main (VOID)
{
    HMQ      hmq;
    FRAMECDATA fcd;
    QMSG     qmsg;

    if (!(hab = WinInitialize(0)))
        return FALSE;

    if (!(hmq = WinCreateMsgQueue(hab, 0)))
        return FALSE;

    /******
    /*  Setup the frame control data for the frame window.          */
    /******
    fcd.cb = sizeof(FRAMECDATA);
    fcd.flCreateFlags = FCF_TITLEBAR      |
                      FCF_SYSMENU       |
                      FCF_MENU          |
                      FCF_SIZEBORDER    |
                      FCF_SHELLPOSITION |
                      FCF_MINMAX        |
                      FCF_TASKLIST;

    fcd.hmodResources = NULLHANDLE;

    /******
    /*  Set our resource key (so PM can find menus, icons, etc).    */
    /******
    fcd.idResources = DDES;

    /******
    /*  Create the frame window.                                     */
    /******
    hFrameWnd = WinCreateWindow(HWND_DESKTOP,
                                WC_FRAME,
                                "DDE Server",
                                0, 0, 0, 0, 0,
                                NULLHANDLE,
                                HWND_TOP,
                                DDES,
                                &fcd,
                                NULL);

    /******
    /*  Verify that the frame was created; otherwise, stop.        */
    /******
    if (!hFrameWnd)
        return FALSE;

    /******
    /*  Set an icon for the frame window.                          */
    /******
    WinSendMsg(hFrameWnd,
               WM_SETICON,
               (MPARAM)WinQuerySysPointer(HWND_DESKTOP,
                                           SPTR_FOLDER,
                                           FALSE),
               NULL);

    /******
    /*  Create a list window child.                                  */
    /******
    hListWnd = WinCreateWindow(hFrameWnd,
                               WC_LISTBOX,
                               NULL,
                               LS_HORZSCROLL,
                               0, 0, 0, 0,
                               hFrameWnd,
                               HWND_BOTTOM,
                               FID_CLIENT,
                               NULL,
                               NULL);

    /******

```

```

/* We must intercept the frame window's messages */
/* (to capture any input from the container control). */
/* We save the return value (the current WndProc), */
/* so we can pass it all the other messages the frame gets. */
/*****
SysWndProc = WinSubclassWindow(hFrameWnd, (PFNWP)LocalWndProc);

WinShowWindow(hFrameWnd, TRUE);

/*****
/* Standard PM message loop - get it, dispatch it. */
/*****
while (WinGetMsg(hab, &qmsg, NULLHANDLE, 0, 0))
{
    WinDispatchMsg(hab, &qmsg);
}

/*****
/* Clean up on the way out. */
/*****
WinDestroyMsgQueue(hmq);
WinTerminate(hab);

return TRUE;
}

/*****
/* LocalWndProc() - window procedure for the frame window. */
/* Called by PM whenever a message is sent to the frame. */
/*****
MRESULT EXPENTRY LocalWndProc(HWND hwnd,ULONG msg,MPARAM mp1,MPARAM mp2)
{
    /* Our inbound DDE stuff */
    PSZ      szClientApp;
    PSZ      szClientTopic;
    PSZ      szReqItem;

    /* Our supported DDE stuff */
    PSZ      szApp      = "DDEdemo";
    PSZ      szTopic    = "System";
    PSZ      szItem     = "Status";
    PSZ      szStatus   = "RUNNING";

    /* System DDE structures */
    CONVCONTEXT context;
    PDDEINIT  pDDEinit;
    PDDESTRUCT pDDEdata;

    /* Miscellaneous */
    PID      pid;
    TID      tid;
    PVOID    mem;

    switch(msg)
    {
        /* All WinDDEInitate calls arrive here */
        case WM_DDE_INITIATE:
            ShowMessage("init");
            hClientWnd = (HWND)mp1;
            pDDEinit = (PDDEINIT)mp2;

            /* Check incoming poll - if the App and Topic match, */
            /* we must acknowledge. If both are NULL, the client is */
            /* searching for anyone - send our names */
            szClientApp = pDDEinit->pszAppName;
            szClientTopic = pDDEinit->pszTopic;
            ShowMessage(szClientApp);
            ShowMessage(szClientTopic);

            if (!strcmpi(szClientApp, szApp) ||
                !strcmpi(szClientApp, NULL))
            {
                if (!strcmpi(szClientTopic, szTopic) ||
                    !strcmpi(szClientTopic, NULL) )
                {
                    context.cb = sizeof(CONVCONTEXT);
                    context.fsContext = 0;
                    WinDdeRespond(hClientWnd,
                                hwnd,

```

```

        szApp,
        szTopic,
        &context);
    }
}
break;

/* Incoming DDE request - get the item name, send the data out. */
case WM_DDE_REQUEST:
    ShowMessage("request in...");
    hClientWnd = (HWND)mp1;

    /* The DDE structure is passed, and */
    /* the client should have shared it with us */
    pDDEdata = (PDDESTRUCT)mp2;
    szReqItem = (BYTE *) (pDDEdata + (pDDEdata->offszItemName));
    ShowMessage(szReqItem);

    /* We support item status, but not anything else */
    if (!strcmpi(szReqItem, szItem))
    {
        ShowMessage("sending...");

        /* Get some sharable memory */
        DosAllocSharedMem((PVOID)&mem,
            NULL,
            sizeof(DDESTRUCT)+21,
            PAG_COMMIT |
            PAG_READ |
            PAG_WRITE |
            OBJ_GIVEABLE);

        /* Get the server's id and give it access */
        /* to the shared memory */
        WinQueryWindowProcess(hClientWnd, &pid, &tid);
        DosGiveSharedMem(&mem, pid, PAG_READ | PAG_WRITE);

        /* Setup DDE data structures */
        /* (11 byte name length, 10 plus NULL, */
        /* 10 byte data length) */
        pDDEdata = (PDDESTRUCT)mem;
        pDDEdata->cbData = 10; /* Data length */
        pDDEdata->fsStatus = 0; /* Status */
        pDDEdata->usFormat = DDEFMT_TEXT; /* Text format */

        /* Go past end of structure for the name */
        pDDEdata->offszItemName = sizeof(DDESTRUCT);

        /* Go past end of structure (and name) for the data */
        pDDEdata->offabData = sizeof(DDESTRUCT)+11;
        strcpy((BYTE *) (pDDEdata + (pDDEdata->offabData)), szStatus);
        WinDdePostMsg(hClientWnd,
            hwnd,
            WM_DDE_DATA,
            pDDEdata,
            DDEPM_RETRY);
    }

else
{
    ShowMessage("rejecting...");
    pDDEdata->cbData = 0; /* Data length */
    pDDEdata->fsStatus = DDE_NOTPROCESSED; /* Status */
    pDDEdata->usFormat = DDEFMT_TEXT; /* Text format */
    WinDdePostMsg(hClientWnd,
        hwnd,
        WM_DDE_ACK,
        pDDEdata,
        DDEPM_RETRY);
}

ShowMessage("sent...");
break;

/* Menu item processing */
case WM_COMMAND:
    switch (SHORT1FROMMP(mp1))
    {
        case IDM_EXIT:
            WinPostMsg(hwnd, WM_CLOSE, 0, 0);

```

```

        break;
    default:
        return (*SysWndProc)(hwnd, msg, mp1, mp2);
        break;
    }
    break;

    /* Send the message to the usual WC_FRAME WndProc */
    default:
        return (*SysWndProc)(hwnd, msg, mp1, mp2);
        break;
    }
    return (MRESULT)FALSE;
}

/*****
/* ShowMessage().
*****/
void ShowMessage(PSZ szText)
{
    WinPostMsg(hListWnd,
               LM_INSERTITEM,
               MPFROMSHORT(LIT_END),
               szText);
}

=====
DDES.RC
=====
#include <os2.h>
#include "ddes.h"

MENU      DDES
BEGIN
    SUBMENU      "Commands",    IDM_MENU
    BEGIN
        MENUITEM "Exit",        IDM_EXIT
    END
END

=====
DDES.H
=====
#define DDES      100
#define IDM_MENU  1000
#define IDM_EXIT  1001

=====
DDES.DEF
=====
NAME      DDES WINDOWAPI

DESCRIPTION 'PM DDE Server Sample'

CODE      MOVEABLE
DATA      MOVEABLE MULTIPLE

STACKSIZE 24576
HEAPSIZE  10240

PROTMODE

=====
DDES.LNK
=====
ddes.obj
ddes.exe
ddes.map

ddes.def

=====
DDES.MAK
=====
CC      =  icc /c /Ge /Gd- /Se /Re /ss /Gm+
LINK    =  link386
HEADERS =  ddes.h

#-----

```



```
# A list of all of the object files.
#-----
ALL_OBJ1 = ddes.obj

all: ddes.exe

ddes.res: ddes.rc ddes.h

ddes.obj: ddes.c $(HEADERS)

ddes.exe: $(ALL_OBJ1) ddes.def ddes.lnk ddes.res
          $(LINK) @ddes.lnk
          rc -p -x ddes.res ddes.exe
```

Entry-Field Controls

An *entry field* is a control window that enables a user to view and edit a single line of text. This chapter describes how to create and use entry-field controls in your PM applications.

About Entry Fields

An entry field provides the text-editing capabilities of a simple text editor and is useful whenever an application requires a short line of text from the user.

If the application requires more sophisticated text-editing capabilities and multiple lines of text from the user, the application can use a multiple-line entry (MLE) field. See *Presentation Manager Programming Guide - Advanced Topics* for more information about MLE controls.

Both the user and the application can edit text in an entry field. Applications typically use entry fields in dialog windows, although they can be used in non-dialog windows as well.

An application creates an entry field by specifying either the WC_ENTRYFIELD window class in the WinCreateWindow function or the ENTRYFIELD statement in a resource-definition file.

Entry-Field Styles

An entry field has a style that determines how it appears and behaves. An application specifies the style in either the WinCreateWindow function or the ENTRYFIELD statement in a resource-definition file. An application can specify a combination of the following styles for an entry field.

Style	Description
ES_ANY	Allows the entry-field text to contain a mixture of double-byte and single-byte characters.
ES_AUTOSCROLL	Automatically scrolls text horizontally to show the insertion point.
ES_AUTOSIZE	Automatically sets the size of the entry field, based on the width of the field's text string and the metrics of the current system font. This style can set the width, height, or both-whichever has a value of -1 in the WinCreateWindow function or resource-definition file. This style affects

	only the initial size of the entry field; it does not adjust the size as the font or text-string width changes.
ES_AUTOTAB	Automatically moves the cursor to the next control window when the user enters the maximum number of characters.
ES_CENTER	Centers text within the entry field.
ES_DBCS	Specifies that the entry-field text consist of double-byte characters only.
ES_LEFT	Left-aligns text within the entry field.
ES_MARGIN	Draws a border around the entry field. The border is 1/2-character wide and 1/4-character high. Without this style, the application draws no border around the entry field. The width of the entry-field rectangle is increased on all sides by the width of this margin. After an entry field with the ES_MARGIN style is created, the WinQueryWindowRect function returns a larger rectangle that includes this margin and whose origin, therefore, is different from the origin specified when the entry field was created. If an application does not adjust for this size difference when moving or sizing an entry field, the entry field becomes larger after each moving and sizing operation.
ES_MIXED	Allows the entry-field text to contain a mixture of single-byte and double-byte characters. Unlike the ES_ANY style, this style lets ASCII DBCS data be converted to EBCDIC DBCS data without causing an overflow condition.
ES_READONLY	Prevents the user from entering or editing text in the entry field.
ES_RIGHT	Right-aligns text within the entry field.
ES_SBCS	Specifies that the entry-field text must consist of single-byte characters only.
ES_UNREADABLE	Displays each character as an asterisk (*). This style is useful when obtaining a password from the user.

Entry-Field Notification Codes

An entry field is always owned by another window. A WM_CONTROL notification message is sent to the owner whenever an event occurs in the entry field. This message contains a notification code that specifies the exact nature of the event. An entry field can send the notification codes described in the following table to its owner.

Notification Code	Description
EN_CHANGE	Indicates that the contents of an entry field have changed.
EN_INSERTMODETOGGLE	Indicates that the insert mode has been toggled.
EN_KILLFOCUS	Indicates that an entry field has lost the keyboard focus.

<code>EN_MEMERROR</code>	Indicates that an entry field cannot allocate enough memory to perform the requested operation, such as extending the text limit.
<code>EN_OVERFLOW</code>	Indicates that either the user or the application attempted to exceed the text limit.
<code>EN_SCROLL</code>	Indicates that the text in an entry field is about to scroll.
<code>EN_SETFOCUS</code>	Indicates that an entry field received the keyboard focus.

An application typically ignores notification messages from an entry field, thereby allowing default text editing to occur. For more specialized uses, an application can use notification messages to filter input. For example, if an entry field is intended for numbers only, an application can use the `EN_CHANGE` notification code to check the contents of the entry field each time the user enters a non-numeric character.

As an alternative, an application can prevent inappropriate characters from reaching an entry field by using `EN_SETFOCUS` and `EN_KILLFOCUS`, in filter code, placed in the main message loop. Whenever the entry field has the keyboard focus, the filter code can intercept and filter `WM_CHAR` messages before the `WinDispatchMsg` function passes them to the entry field. An application also can respond to certain keystrokes, such as the Enter key, as long as the entry-field control has the keyboard focus.

Default Entry-Field Behavior

The following table lists and describes all the messages specifically handled by the predefined entry-field control-window class (`WC_ENTRYFIELD`).

Message	Description
<code>EM_CLEAR</code>	Deletes the current text selection from the control window.
<code>EM_COPY</code>	Copies the current text selection to the system clipboard, in <code>CF_TEXT</code> format.
<code>EM_CUT</code>	Copies the current text selection to the system clipboard, in <code>CF_TEXT</code> format, and deletes the selection from the control window.
<code>EM_PASTE</code>	Copies the current contents of the system clipboard that have <code>CF_TEXT</code> format, replacing the current text selection in the control window.
<code>EM_QUERYCHANGED</code>	Returns <code>TRUE</code> if the text has changed since the last <code>EM_QUERYCHANGED</code> message.
<code>EM_QUERYFIRSTCHAR</code>	Returns the offset to the first character visible at the left edge of the control window.
<code>EM_QUERYREADONLY</code>	Determines whether the entry field is in the read-only state.
<code>EM_QUEYSSEL</code>	Returns a long word that contains the offsets for the first and last characters of the current selection in the control window.
<code>EM_SETFIRSTCHAR</code>	Scrolls the text so that the character at the specified offset is the first character visible at the left edge of the control window.
<code>EM_SETINSERTMODE</code>	Toggles the text-entry mode between

	insert and overstrike.
EM_SETREADONLY	Sets the entry field to the read-only state.
EM_SETSEL	Sets the current selection to the specified character offsets.
EM_SETTEXTLIMIT	Allocates memory from the control heap for the specified maximum number of characters, returning TRUE if it is successful and FALSE if it is not. Failure causes the entry field to send a WM_CONTROL message with the EN_MEMERROR notification code to the owner window.
WM_ADJUSTWINDOWPOS	Changes the size of the control rectangle if the control has the ES_MARGIN style.
WM_BUTTON1DBLCLK	Occurs when the user presses mouse button 1 twice.
WM_BUTTON1DOWN	Sets the mouse capture and keyboard focus to the entry field, and prepares to track the movement of the mouse during WM_MOUSEMOVE messages.
WM_BUTTON1UP	Releases the mouse.
WM_BUTTON2DOWN	Returns TRUE to prevent this message from being processed further.
WM_BUTTON3DOWN	Returns TRUE to prevent this message from being processed further.
WM_CHAR	Handles text entry and other keyboard input events.
WM_CREATE	Validates the requested style and sets the window text.
WM_DESTROY	Frees the memory used for the window text.
WM_ENABLE	Sent when an application changes the enabled state of a window.
WM_MOUSEMOVE	If the mouse button is down, the entry field tracks the text selection. If the mouse button is up, the entry field sets the mouse pointer to the default arrow shape.
WM_PAINT	Draws the entry field and text.
WM_QUERYDLGCODE	Returns the predefined DLGC_ENTRYFIELD constant.
WM_QUERYWINDOWPARAMS	Returns the requested window parameters.
WM_SETFOCUS	If the entry field is gaining the focus, it creates a cursor and sends the owner window a WM_CONTROL message with the EN_SETFOCUS notification code. If the entry field is losing the focus, it destroys the current cursor and sends the owner window a WM_CONTROL message with the EN_KILLFOCUS notification code.
WM_SETSELECTION	Toggles the current selection status.
WM_SETWINDOWPARAMS	Sets the specified window parameters, redraws the entry field, and sends the owner window a WM_CONTROL message with the EN_CHANGE notification code.
WM_TIMER	Blinks the insertion point if the entry field has the focus. The entry field

scrolls the text, if necessary, while
extending the selection to text that
becomes visible in the window.

Entry-Field Text Editing

The user can insert (type) text or numeric values in an entry field when that entry field has the keyboard focus. An application can insert text by using the `WinSetWindowText` function. An application can insert numeric values by using the `WinSetDlgItemShort` function. The text or numeric value is inserted into the entry field at the cursor position.

The entry field's entry mode, either insert or overstrike, determines what happens when the user enters text. The user sets the entry mode by pressing the Insert key; the entry mode toggles each time the Insert key is pressed. The application can set the entry mode by sending the `EM_SETINSERTMODE` message to the entry field.

The cursor position, identified by a blinking bar, is specified by a character offset relative to the beginning of the text. The user can set the cursor position by using the mouse or the Arrow keys. An application can set the cursor position by using the `EM_SETSEL` message. This message directs the entry field to move the blinking bar to the given character position.

The `EM_SETSEL` message also sets the selection. The selection is one or more characters of text on which the entry field carries out an operation, such as deleting or copying to the clipboard. The user selects text by pressing the Shift key while moving the cursor, or by pressing mouse button 1 while moving the mouse. An application selects text by using the `EM_SETSEL` message to specify the cursor position and the anchor point. The selection includes all text between the cursor position and the anchor point. If the cursor position and anchor point are equal, there is no selection. An application can retrieve the selection (cursor position and anchor point) by using the `EM_QUERYSEL` message.

The user can delete characters, one at a time, by pressing the Delete key or the Backspace key. The Delete key deletes the character to the right of the cursor; the Backspace key deletes the character to the left of the cursor. The user also can delete a group of characters by selecting them and pressing the Delete key. An application can delete selected text by using the `EM_CLEAR` message.

An application can use the `EM_QUERYCHANGED` message to determine whether the contents of an entry field have changed.

An application can prevent the user from editing an entry field by setting the `ES_READONLY` style in the `WinCreateWindow` function or in the `ENTRYFIELD` statement in the resource-definition file. The application also can set and query the read-only state by using the `EM_SETREADONLY` and `ES_QUERYREADONLY` messages.

If text extends beyond the left or right edges of an entry field, the user can scroll the text by using the Arrow keys. An application can scroll the text by using the `EM_SETFIRSTCHAR` message to specify the first character visible at the left edge of the entry field. For scrolling to occur, the entry field must have the `ES_AUTOSCROLL` style. An application can use the `EM_QUERYFIRSTCHAR` message to obtain the first character that is currently visible.

Entry-Field Control Copy and Paste Operations

The user can cut, copy, and paste text in an entry field by using the Shift+Delete and Ctrl+Insert key combinations. An application, either by itself or in response to the user, can cut, copy, and paste text by using the `EM_CUT`, `EM_COPY`, and `EM_PASTE` messages. An application can use the `ES_CUT` and `EM_COPY` messages to copy the selected text to the clipboard. The `EM_CUT` message also deletes the text (`EM_COPY` does not). The `EM_PASTE` message copies the text on the clipboard to the current position in the entry field, replacing any existing text with the copied text. An application can delete the selected text, without copying it to the clipboard, by using the `EM_CLEAR` message.

Entry-Field Text Retrieval

An application can retrieve selected text from an entry field by calling `WinQueryWindowText` and then sending an `EM_QUERYSEL` message to retrieve the offsets to the first and last characters of the text selection. These offsets are used to retrieve selected text.

An application can retrieve numeric values by calling `WinQueryDlgItemShort`, passing the entry-field identifier and the handle of the owner window. `WinQueryDlgItemShort` converts the entry-field text to a signed or unsigned integer and returns the value in a specified variable. The application can use the `WinWindowFromID` function to retrieve the handle of the control window. The entry-field identifier is specified in the dialog template in the application's resource-definition file.

Using Entry-Field Controls

This section explains how to perform the following tasks:

- Create an entry field in a dialog or client window
- Change the default size of the entry field

Creating an Entry Field in a Dialog Window

A dialog window usually serves as the parent and owner of an entry field. The dialog window often includes a button that indicates whether the user wants to carry out an operation. When the user selects the button, the application queries the contents of the entry field and proceeds with the operation.

The definition of an entry field in an application's resource-definition file sets the initial text, window identifier, size, position, and style of the entry field. The following example shows how to define an entry field as part of a dialog template:

```
DLGTEMPLATE IDD_SAMPLE
BEGIN
    DIALOG "Sample Dialog", ID_DLG, 7, 7, 253, 145, FS_DLGBOARDER,0
    BEGIN
        DEFPUSHBUTTON "~OK", DID_OK, 8, 151, 50, 23, WS_GROUP
        ENTRYFIELD "Here is some text", ID_ENTFLD, 42, 46, 68, 15,
            ES_MARGIN | ES_AUTOSCROLL
    END
END
```

Creating an Entry Field in a Client Window

To create an entry field in a non-dialog window, an application calls `WinCreateWindow` with the window class `WC_ENTRYFIELD`. The entry field is owned by an application's client window, whose window procedure receives notification messages from the entry field.

The following code fragment shows how to create an entry field in a client window:

```
#define ID_ENTRYFIELD 5

HWND hwnd, hwndEntryField1, hwndClient;
LONG xPos = 50, yPos = 100;
LONG xWidth = 100, yHeight = 20;

hwndEntryField1 = WinCreateWindow(
    hwndClient, /* Parent-window handle */
    WC_ENTRYFIELD, /* Window class */
    "initial text", /* Initial text */
    WS_VISIBLE | /* Visible when created */
    ES_AUTOSCROLL | /* Scroll text */
```

```

ES_MARGIN,          /* Create a border      */
xPos, yPos,         /* x and y position    */
xWidth, yHeight,    /* Width and height    */
hwnd,              /* Owner-window handle */
HWND_TOP,          /* Z-order position    */
ID_ENTRYFIELD,     /* Window identifier   */
NULL,              /* No control data     */
NULL);             /* No pres. parameters */

```

Changing the Default Size of an Entry Field

The default text limit of an entry field is 32 characters. An application can set a non-default size when creating an entry field by setting the *cchEditLimit* member of an ENTRYFDATA structure and supplying a pointer to the structure as the *pCuiData* parameter to WinCreateWindow.

The following code fragment creates an entry field with a text limit of 12 characters:

```

HWND hwndEntryField2;
HWND hwndClient;
ENTRYFDATA efd;
LONG xPos      = 50,
     yPos      = 50;
LONG xWidth    = -1,
     yHeight   = -1;    /* Must be -1 for ES_AUTOSIZE */

/* Initialize the ENTRYFDATA structure */
efd.cb = sizeof(ENTRYFDATA);
efd.cchEditLimit = 12;
efd.ichMinSel = 0;
efd.ichMaxSel = 0;

/* Create the entry field */
hwndEntryField2 = WinCreateWindow(
    hwndClient,          /* Parent-window handle */
    WC_ENTRYFIELD,      /* Window class          */
    "projects.xls",     /* No initial text       */
    WS_VISIBLE |        /* Visible when created  */
    ES_MARGIN |         /* Create a border       */
    ES_AUTOSCROLL |     /* Scroll text           */
    ES_AUTOSIZE,        /* System sets the size  */
    xPos, yPos,         /* x and y positions     */
    xWidth, yHeight,    /* Width and height      */
    hwndClient,         /* Owner-window handle   */
    HWND_TOP,          /* Z-order position     */
    0,                 /* Window identifier     */
    &efd,              /* Control data          */
    NULL);             /* No pres. parameters  */

```

To expand or reduce the text limit after creating the entry field, an application can send an EM_SETTEXTLIMIT message specifying a new maximum text limit for the entry field. The following code fragment increases to 20 characters the text limit of the entry field created in the previous example:

```
WinSendMsg(hwndEntryField2, EM_SETTEXTLIMIT, (MPARAM)20, (MPARAM)0);
```

Retrieving Text From an Entry Field

An application can use the WinQueryWindowTextLength and WinQueryWindowText functions to retrieve the text from an entry field.

WinQueryWindowTextLength returns the length of the text; WinQueryWindowText copies the window text to a buffer.

Typically, an application needs to retrieve the text from an entry field only if the user changes the text. An entry field sends an EN_CHANGE notification code in the low word of the first message parameter of the WM_CONTROL message whenever the text changes. The following code fragment sets a flag when it receives the EN_CHANGE code, checks the flag during the WM_COMMAND message and, if it is set, retrieves the text of the entry field:

```
HWND hwnd;
ULONG msg;
MPARAM mparam;
CHAR chBuf[64];
HWND hwndEntryField;
LONG cbTextLen;
LONG cbTextRead;
static BOOL fFieldChanged = FALSE;

switch (msg) {
    case WM_CONTROL:
        switch (SHORT1FROMMP(mparam)) {
            case IDD_ENTRYFIELD:

                /* Check if the user changed the entry-field text. */
                if ((USHORT) SHORT2FROMMP(mparam) == EN_CHANGE)
                    fFieldChanged = TRUE;
                return 0;
        }

    case WM_COMMAND:
        switch (SHORT1FROMMP(mparam)) {
            case DID_OK:

                /* If the user changed the entry-field text, */
                /* obtain the text and store it in a buffer. */
                if (fFieldChanged) {
                    hwndEntryField = WinWindowFromID(hwnd,
                        IDD_ENTRYFIELD);
                    cbTextLen = WinQueryWindowTextLength(hwndEntryField);
                    cbTextRead = WinQueryWindowText(hwndEntryField,
                        sizeof(chBuf), chBuf);

                    . /* Do something with the text. */
                    .
                }
                WinDismissDlg(hwnd, 1);
                return 0;
        }
}
```

File Dialog Controls

File dialog controls provide basic functions that enable users to do the following:

- Display and select from a list of drives, directories, and files
- Enter a file name directly
- Filter the file names before they are displayed
- Display active network connections
- Specify .TYPE EA extended attributes
- Interact with a single-selection or multiple-selection file dialog
- Interact with a modal or modeless file dialog

These basic functions can be extended to meet the requirements of PM applications.

About File Dialog Controls

The file dialog control enables you to implement *Open* or *SaveAs* dialogs.

Customizing the File Dialog

You can customize the File Dialog control by using the standard controls and adding any of your own design. Specify a standard control by including the control name, ID, and style in the dialog.

Using File Dialog Controls

This section describes how to create:

- A file dialog
 - An Open dialog
 - A SaveAs dialog
-

Creating a File Dialog

To present a file dialog to users, your application must do the following:

1. Allocate storage for a FILEDLG data structure and set all fields to NULL.
2. Initialize the fields in the FILEDLG data structure.

The application must do the following:

- a. Set the *cbSize* field to the size of the data structure.
- b. Set the *//* field to indicate the type of dialog. You must set the FDS_OPEN_DIALOG or FDS_SAVEAS_DIALOG flags.

The application can set the following:

- a. An application-specific title. Pass the pointer to a null-terminated string in the *pszTitle* field.
 - b. An application-specific text for the *OK* push button. Pass the pointer to a null-terminated string in the *pszOKButton* field.
 - c. A custom dialog procedure to provide application-specific function. Pass the pointer to a window procedure in the *pfnDlgProc* field.
 - d. Set other FDS_* flags in the *//* field to customize the dialog style.
 - e. Pass the initial position of the dialog in the *x* and *y* fields.
3. Initialize the FILEDLG data structure with any values that users should see when they invoke the dialog for the first time. For example, you can:
 - a. Pass the name of the first drive from which file information will be displayed in the *pszDrive* field.
 - b. If you want to limit user selections, pass a list of drives from which the user can choose in the *papszDriveList* field. Otherwise, the system defaults to showing all available drives.
 - c. Pass the name of an extended-attribute filter to be used to filter file information in the *psz/Type* field.

- d. Pass a list of extended attributes in the *papsz/TypeList* field. By selecting from this list, users can filter file information.
 - e. Pass the name of the initial file to be used by the dialog in the *szFullFile* field. This can be a file name or a string filter, such as *.dat, to filter the initial file information. This field can be fully qualified to select the initial drive and directory.
4. Invoke the file dialog. Call WinFileDlg and pass the dialog's owner window handle and a pointer to the initialized FILEDLG data structure.
5. Verify the return value from WinFileDlg. If it is successful, the application can create the file dialog (either Open or SaveAs) by using the file name or file names returned from the dialog.

Creating an Open Dialog

When the Open dialog is invoked, the fields in the dialog box are updated with the fields passed in the FILEDLG data structure. The values passed in the *szFullFile* field of the data structure are displayed in the *File Name* field, the Directory list box, and the *Drive* field. The value passed in the *pszType* field is displayed in the *Type* field.

Creating a SaveAs Dialog

The SaveAs dialog is identical to the Open dialog with these exceptions:

- By default, the file names in the file list box are grayed and cannot be selected, although the list box can be scrolled.
- When the user clicks on the *OK* push button or presses the Enter key, the file name in the *File Name* field is passed to the application, and the application saves, rather than opens, the file.
- The titles of the file name, filter, and dialog are SaveAs rather than Open.

Graphical User Interface Support for File Dialog Controls

This section provides information about the file dialog user interface.

Name Field

The *File Name* field is a single-line entry (SLE) field used to display the name of a file that was selected from the file list box or entered directly by the user. As the user types, the file or files matching the user entry are scrolled into view in the file list box. The first file name that most closely matches the file name typed by the user is placed at the top of the list box. When the user types a character that causes a mismatch, the file at the top of the list is displayed.

When the user presses the Enter key, the dialog returns the selected file name to the application. The application then initiates the default action of opening the file. When a file name is not valid, such as when the file does not exist, the application displays an error message.

The *File Name* field displays the currently selected file name or the current string filter. When a filter is specified in the *szFullFile* field of the FILEDLG data structure, the string filter is displayed without the path information. The string filter remains in the field until a file is selected or the user types over the data in the field.

When a file name is not specified, the *File Name* field is blank.

File List Box

The File list box is a single- or multiple-selection list box that is scrollable both horizontally and vertically. It contains all the files that meet the filter criteria, sorted by name.

When the file dialog is a single-selection dialog, the selected file name is placed in the *File Name* field. When the file dialog is a multiple-selection dialog, the topmost selected file name is placed in the *File Name* field. When the user double-clicks on a file name, the dialog exits and returns the selected file or files to the application for opening.

Directory List Box

The Directory list box is a single-selection list box that is scrollable both horizontally and vertically.

The Directory list box displays the path in the *szFullFile* field of the FILEDLG data structure as a list of each parent subdirectory. Any subdirectories of the selected directory also are displayed. Each directory level is indented to show the path, and the current working directory level is indicated by an arrow. The top entry is always the root directory, with the drive specification preceding it. When the *szFullFile* field is NULL, the current path of the current drive is displayed. The user selects a new subdirectory by double-clicking on the subdirectory name. This action updates the Directory list box.

Drive Field

The *Drive* field contains a drop-down list of the logical drives. This field cannot be edited by the user.

The *Drive* field displays the value passed in the *papszDriveList* field of the FILEDLG data structure. If the application does not specify a drive list, all drives currently available on the system are displayed. When the drop-down list is displayed, the current drive is highlighted. When the user selects a drive, the display is refreshed. When either the user-specified drive or the default drive has a volume label, the volume label is displayed also.

Users can access networked files by associating logical disks with remote servers, or they can enter the name and ID of the server in the *File Name* field. When the server name entered is not found in the Drive drop-down list, it is added to the list and displayed in the *Drive* field.

Type Field

The *Type* field contains a drop-down list of extended-attribute filters.

The *Type* field displays the value passed in the *pszType* field of the FILEDLG data structure. The current setting is highlighted when the drop-down list is displayed.

When a type filter is not specified by the application, <All Files> is displayed and no extended-attribute type filtering is used with the initial display.

All files affected by the string filter and the extended-attribute type filter criteria are displayed, based on how the filters are to be used. The default is that all file names meeting the intersection of the two filters are shown. When users change the value in the *Type* field, the File list box is updated to display a list of files that meet the new type filter criteria. Files that meet both the string filter and extended-attribute type

filter are displayed.

Standard Push Button and Default Action

The *OK* push button initiates the default action.

When a subdirectory is selected, the *File Name* field is empty. When the user clicks on the *OK* push button or presses the Enter key, the subdirectory is opened and the displayed values in the File list box and the Directory list box are refreshed.

When a file name is selected, selection of subdirectories is canceled and the *File Name* field is updated with the name of the selected file. When the user clicks on the *OK* push button or presses the Enter key, the file displayed in the *File Name* field is returned to the application for opening.

Subclassing the Default File Dialog Procedure

The name of the dialog procedure is assigned to the *pfnDlgProc* field of the FILEDLG data structure.

Font Dialog Controls

Font dialog controls provide basic functions that give users the ability to display and select from a list of:

- Font family names installed on the system
- Available styles for each font
- Available sizes for each font
- Emphasis styles available for each font

Users can view their selections, using a sample character string in a preview area, and interact with a modal or modeless font dialog. This chapter explains how font dialog controls can be extended to meet the requirements of PM applications.

About Font Dialog Controls

In the font dialog control, *family face* is defined as the name of the typeface. Courier, Times New Roman**, and Helvetica** are examples of commonly used family faces. Type styles include normal, **bold**, *italic*, and ***bold italic***. *Size* is the point size, or vertical measurement, of the type. Font emphasis styles include outline, underline, and strikeout.

Customizing the Font Dialog

You can create a font dialog by customizing the font dialog control, using the standard controls and adding any controls of your own design. Specify a standard control by including a control of the same class, ID, and style as in the font dialog.

The minimum set of controls required for the font dialog are:

- DID_CANCEL_BUTTON
- DID_DISPLAY_FILTER

- DID_NAME
- DID_OK_BUTTON
- DID_OUTLINE
- DID_PRINTER_FILTER
- DID_SAMPLE
- DID_SIZE
- DID_STRIKEOUT
- DID_STYLE
- DID_UNDERSCORE

Even if your dialog does not use all of the required controls, you must include them. You can make the unused controls invisible so that your application users are not confused.

Using Font Dialog Controls

This section describes how to create a font dialog.

Creating a Font Dialog

To present a font dialog to users, your application must do the following:

1. Allocate storage for a FONTDLG data structure and set all fields to NULL.
2. Initialize the fields in the FONTDLG data structure.

The application must do the following:

- a. Set the *cbSize* field to the size of the data structure.
- b. Set either the *hpsScreen* or the *hpsPrinter* presentation space field, or both. You must have a valid presentation space from which to query fonts.
- c. Pass the pointer to a buffer in which to return the family name selected (*pszFamilyname*) and the size of the buffer (*usFamilyBufLen*). If the application requires a default font, pass the family name of the font in this buffer. When the first character in *pszFamilyName* is NULL, no family name is initially selected and the dialog defaults to the system font.

The application can choose to set the following:

- a. An application-specific title. Pass the pointer to a null-terminated string in the *pszTitle* field.
 - b. An application-specific preview string. Pass the pointer to a null-terminated string in the *pszPreview* field.
 - c. Application-specific available font sizes for outline fonts. Pass the pointer to a null-terminated string containing point sizes, separated by spaces in the *pszPtSizeList* field.
 - d. A custom dialog procedure to provide application-specific function. Pass the pointer to a window procedure in the *pfnDlgProc* field.
 - e. Set the appropriate FNTS_* flags in the *fl* field to customize the dialog style.
 - f. Set the FNTF_NOVIEWPRINTERFONTS or FNTF_NOVIEWSCREENFONTS flags to customize the dialog style when working with printer fonts in the *flFlags* field. These filter flags should be initialized only when both the *hpsScreen* and the *hpsPrinter* presentation space fields are non-NULL.
 - g. Pass the initial position of the dialog in the *x* and *y* fields.
3. Initialize the FONTDLG data structure with any values that users should see when they invoke the dialog for the first time.

For example, you can pass the following information about the font in these fields of the FONTDLG structure:

Field	Font characteristic
-------	---------------------

clrBack	Font background color.
clrFore	Font foreground color.
fl	Font flags.
flStyle	Style bits.
fxPointSize	Point size of the font.
flType	Selected type bits.
pszFamilyname	Family name of the font.
usWeight	Font weight.
usWidth	Font width.

The following code fragment shows how to set the fields in the FONTDLG structure:

```

FONTDLG fd;                                /* Font dialog structure */
strcpy(szCurrentFont,"Tms Rmn");           /* Times Roman font */
memset(&fd,                                /* Storage of the */
      0,                                  /* FONTDLG structure */
      sizeof(FONTDLG));

fd.cbSize=sizeof(FONTDLG);                 /* Size of structure */
fd.pszFamilyname=szCurrentFont;            /* Initial font type */
fd.fxPointSize=MAKEFIXED(24,0);            /* Initial font to 24 point */
fd.usFamilyBufLen=FACE_SIZE;               /* Family buffer length */
fd.clrFore=SYSCLR_WINDOWTEXT;              /* Foreground color */
fd.clrBack=SYSCLR_WINDOW;                  /* Background color */
fd.fl=FNTS_CENTER |                        /* Font flags */
      FNTS_INITFROMATTRS;

```

4. Invoke the font dialog. Call WinFontDlg and pass the dialog's parent window handle, owner window handle, and a pointer to the initialized FONTDLG data structure.
5. Check the return value from WinFontDlg. If it is successful, the selected font can be used by the application. The information returned in the *Attrs* field of the FONTDLG data structure is used.

Graphical User Interface Support for Font Dialog Controls

This section contains information about the graphical user interface support.

Name Field

The *Name* field is a drop-down list that displays a font family name. When the font dialog is invoked, the value displayed in this field is either an application-supplied family name or the default system font.

When users select a family name from the drop-down list, the *Name* field display is refreshed with the selected family name. The preview area is updated to show the sample character string in the selected family face, using the font style, size, and emphasis currently in effect.

Style Field

The *Style* field is a drop-down list that displays a font style. When the font dialog is invoked, the value displayed in this field is either an application-specified font style or the system default.

When users select a font style from the drop-down list, the *Style* field display is refreshed with the selected style name. The preview area is updated to show the sample character string in the selected font style, using the family name, size, and emphasis currently in effect.

Size Field

The *Size* field is a drop-down combination box that displays available font sizes. Users can display and select from a list of available sizes for a font, or they can type a font size directly into the entry field.

When users select a font size from the drop-down list, the *Size* field display is refreshed with the selected size. The preview area is updated to show the character string in the selected font size, using the family name, font style, and emphasis currently in effect.

The font sizes included in the drop-down list are dependent on the character definition of the font. For image or raster fonts, all available sizes are listed. For outline fonts, the default sizes are 8, 10, 12, 14, 18, and 24 points. If required, the application can specify the available sizes for outline fonts.

When users type a font size in the entry field, the preview area is updated immediately. The *Size* field will accept a fixed point number, such as 24.25, with up to four places saved after the decimal.

Emphasis Group Box

The *Emphasis group box* is a multiple-selection field that contains a list of emphasis styles (*Outline*, *Underline*, *Strikeout*) available for each font.

When users select an emphasis style, the preview area is updated immediately. The Outline selection is not available for image fonts.

Preview Area

The *Preview* area enables users to view their font family, style, size and emphasis selections as they make them. It contains a sample character string that is defined by the application. The default character string is abcdABCD. The Preview area displays font sizes as large as 48 points. As the size of the font increases, the sample displayed is clipped by the borders of the area.

Filter Check Box

The *Filter* check box enables users to limit the font family name drop-down list to select from fonts that are displayable only, printable only, or a merged list. The initial setting of the *Filter* check box is specified by the application.

Standard Push Button and Default Action

The dialog can be dismissed with either the **OK** or **Cancel** push buttons.

Subclassing the Default Font Dialog Procedure

The name of the dialog procedure is assigned to the *pfnDlgProc* field of the FONTDLG data structure.

Frame Windows

A *frame window* is the basic window used by most Presentation Manager applications to enable the user to perform manipulation functions. This chapter explains how to create and use frame windows in PM applications.

About Frame Windows

An application nearly always starts with a frame window to create a *composite window* (for example, a main window) that consists of the frame window, several frame-control windows, and a client window. The frame controls conform to the Common User Access (CUA) user interface guidelines. The frame window coordinates the actions of the frame controls and client window, enabling the composite window to act as a single unit.

Frame windows have the preregistered public window class WC_FRAME. The frame-window class, like the preregistered control classes, defines the appearance and behavior of the frame window.

Main Window

The *main window* of an application, typically, is composed of a frame window and a client window. The frame window usually includes control windows such as a title bar, system menu, menu bar (*action bar* or *menu* in user terminology), and scroll bars.

A frame window provides the standard services the user expects from a window—for example, moving, sizing, minimizing, and maximizing. The frame window receives input from the control windows (called *frame controls*) and sends messages to both the frame controls and the client window.

Frame Controls

When creating a frame window, an application also can create one or more frame controls as child windows of the frame window. Most frame windows contain at least a system menu and title bar. Other optional controls might include a menu bar and scroll bar as shown above.

An application can create a frame window with specified frame controls by calling WinCreateStdWindow with the appropriate frame-control flags.

The frame window owns the child frame-control windows, which can send notification messages that tell the frame window what the user is doing with the frame controls. For example, using a mouse, a user can move a window by clicking the title bar and dragging the window to a new position. The title-bar control responds to the click by sending a message to the frame window, notifying it of the user's request to move the window. Then the frame window tracks the mouse motion and moves the frame window and all of its child windows to the new position.

PM, rather than the application, handles the processing of the frame controls, thus providing the user a consistent interface for manipulating and interacting with windowed applications on the screen. Frame controls are described in individual chapters. For more information about control windows, see [Control Windows](#).

Client Window

Every main window has a *client window*, which is the window in which the application displays output and receives mouse and keyboard input from the user. What an application displays in the client window, how it displays it, and how it interprets input to the window are controlled by the client's application-defined window procedure.

An application creates the client window when it creates the frame window. The client window, which is specific to the application, is nearly always created using a *private window class* (a class registered by the application). Like a frame control, the client window is a child window and is owned by the frame window. This means, for example, that the client window is moved when the frame window moves, is clipped to the frame-window size, and is destroyed when the frame window is destroyed.

The relationship between the frame window and the client window allows the frame window to pass messages between other frame controls and the client window. For example, a client window can send a message to the frame window requesting that the frame window change the window title. The frame window, in turn, sends a message to the title-bar control, telling it to change the title of the window.

Additional Frame-Window Items

In addition to its frame controls, a frame window also can contain a sizing border and the minimize and maximize buttons (also known as minimize and maximize *icons*). These items are not frame controls, because the frame window draws and maintains them. (*Frame controls* are windows that draw and maintain themselves.)

The sizing border encloses the frame window and lets the user change the size of the window using a mouse. The minimize button, at the right end of the title bar, lets the user reduce the frame window to an icon. The maximize button, to the right of the minimize button, lets the user enlarge the window so that it fills the screen. An application can add these items to a frame window by using the FCF_SIZEBORDER, FCF_MAXBUTTON, and FCF_MINBUTTON (or FCF_MINMAX) styles. (The FCF_MINMAX style adds both a maximize button and a minimize button.)

Frame-Control Identifiers

A frame window uses a set of standard constants to identify the frame controls and the client window. The *frame-control identifiers* all begin with the prefix FID_ and can be used in functions such as WinWindowFromID to uniquely identify a given control or the client window. The frame controls also use these identifiers in notification messages sent to the frame window. The following table describes the frame-control identifiers:

Identifier	Description
FID_CLIENT	Identifies a client window.
FID_HORZSCROLL	Identifies a horizontal scroll bar.
FID_MENU	Identifies a menu.
FID_MINMAX	Identifies the minimize and maximize (<i>window-sizing</i>) buttons.
FID_SYSMENU	Identifies a system menu.
FID_TITLEBAR	Identifies a title bar.
FID_VERTSCROLL	Identifies a vertical scroll bar.

Frame-Window Creation

An application typically creates a frame window by using `WinCreateStdWindow`, which creates a frame window, a client window, and the specified frame controls. The application also can call `WinCreateWindow` with the `WC_FRAME` window class, which creates the frame window and controls but not the client window. To create the client, the application can call `WinCreateWindow`, specifying the original frame window as the parent and owner.

An application also can use a frame window to create a dialog window. For a dialog window, the frame window contains control windows but no client window. The application creates the dialog window by using `WinLoadDlg` or `WinCreateDlg`. These functions require an appropriate dialog template from the application's resource-definition file. The dialog template specifies the styles and dimensions for the frame window and for the control windows that compose the dialog window.

Frame Window Controls and Styles

An application uses frame-control flags in `WinCreateStdWindow` to specify which frame controls to give to the frame window. Frame-control flags are constants that have the `FCF_` prefix.

The frame-window class (`WC_FRAME`), like other public window classes, provides many class-specific window styles that applications can use to adapt the appearance and behavior of a frame window. To specify the frame-window styles, an application can use either frame-control flags or the frame-window style constants, which have the `FS_` prefix. Each style constant has a corresponding frame-control flag. Both produce exactly the same styles in a frame window. Typically, if an application is creating a frame window that uses frame controls, the application uses frame-control flags to specify the frame-window styles-if not, the application uses frame-style constants. An application can combine the frame-style constants with the standard window styles when creating a frame window.

When an application calls `WinCreateStdWindow` without setting any frame-control flags, the function creates a standard window that is invisible and behind all its sibling windows, has a width and height of 0, and is positioned at the lower-left corner of its parent window. After the call to `WinCreateStdWindow` returns, the application can use `WinSetWindowPos` to change the window's size, coordinates, z-order position, and visibility.

If an application calls `WinCreateStdWindow` with the `FCF_SHELLPOSITION` frame-control flag, the function creates the window so that it is in front of its sibling windows and has a standard size and coordinates determined by the system.

Frame-Window Resources

If an application specifies `FCF_ACCELTABLE`, `FCF_ICON`, `FCF_MENU`, `FCF_STANDARD`, `FS_ACCELTABLE`, `FS_ICON`, or `FS_STANDARD` when creating a frame window, the application must provide the resources to support the specified style. Failure to do so causes the window creation to fail. Depending on the style, a frame window might attempt to load one or more resources from the application's executable files.

The following table shows the frame-control flags and frame-window styles that require resources:

Flag	Style	Description
<code>FCF_ACCELTABLE</code>	<code>FS_ACCELTABLE</code>	Requires an accelerator-table resource. The frame window uses the accelerator table to translate <code>WM_CHAR</code> messages to <code>WM_COMMAND</code> , <code>WM_HELP</code> , or <code>WM_SYSCOMMAND</code> messages.
<code>FCF_ICON</code>	<code>FS_ICON</code>	Requires an icon resource. The frame window draws the icon when the user minimizes the

		window.
FCF_MENU	FS_MENU	Requires a menu-template resource. A frame window uses the menu template to create a menu containing the commands and menus specified by the resource.
FCF_STANDARD	FS_STANDARD	Requires a menu-template resource (FCF_STANDARD only), an accelerator-table resource, and an icon resource.

You can use the resource compiler to add icon, menu, and accelerator-table resources to the application's executable file. Each resource must have a resource identifier that matches the resource identifier specified in the FRAMECDATA structure passed to WinCreateWindow or in the *idResources* parameter of WinCreateStdWindow.

Note: For detailed information about icon, menu, and accelerator-table resources, see [Mouse Pointers and Icons](#), [Menus](#), and [Keyboard Accelerators](#), respectively.

The following sample code illustrates how to use WinCreateStdWindow to load and set up certain resources for a frame window. Normally the first step is to set up a header file defining the the IDs of the applicable resources:

```
#define ID_RESOURCE 301

#define IDM_OPTIONS 350
#define IDM_SHIFT 351
#define IDM_EXIT 352
```

Then, make a resource (.RC) file, defining each resource:

```
#include <os2.h>

/* Icon */
POINTER ID_RESOURCE sampres.ico

/* Accelerator table */
ACCELTABLE ID_RESOURCE
BEGIN
    VK_F10,    IDM_SHIFT,    VIRTUALKEY
    VK_F3 ,    IDM_EXIT,    VIRTUALKEY
END

/* Menu */
MENU ID_RESOURCE
BEGIN
    SUBMENU "~Options", IDM_OPTIONS
    BEGIN
        MENUITEM "~Shift Colors\tF10", IDM_SHIFT
        MENUITEM "~Exit\tF3", IDM_EXIT
    END
END
```

When using WinCreateStdWindow with more than one resource, each resource can have the same ID, as in the above example (ID_RESOURCE or 1), *but only if each resource is of a different type*. Resources of the same type must have unique IDs. Use FCF flags to indicate what resources to load:

```
ULONG flFrameFlags=
    FCF_TITLEBAR      | /* Title bar          */
    FCF_SIZEBORDER    | /* Size border        */
    FCF_MINMAX        | /* Min & Max buttons  */
```

```

FCF_SYSMENU      | /* System menu          */
FCF_SHELLPOSITION | /* System size & position */
FCF_TASKLIST     | /* Add name to task list  */
FCF_ICON         | /* Add icon               */
FCF_ACCELTABLE   | /* Add accelerator table  */
FCF_MENU;        | /* Add menu               */

```

Use 0 (or NULL) in the seventh parameter of WinCreateStdWindow to indicate that the resource is stored in the application file, as follows:

```

hwndFrame = WinCreateStdWindow(
    HWND_DESKTOP,      /* Parent is desktop window */
    WS_VISIBLE,        /* Make frame window visible */
    &flFrameFlags,      /* Frame controls            */
    "ResSamClient",    /* Window class for client   */
    NULL,              /* No window title           */
    WS_VISIBLE,        /* Make client window visible */
    (HMODULE) 0,        /* Resources in application module */
    ID_RESOURCE,       /* Resource identifier        */
    NULL);             /* Pointer to client window handle */

```

Following is the full listing of the sample program:

```

#define INCL_PM
#include <os2.h>

MRESULT EXPENTRY ClientWndProc(HWND hwnd,ULONG msg,MPARAM mp1,MPARAM mp2);

int main(int argc, char *argv, char *envp)
{
    HWND hwndFrame;
    HWND hwndClient;
    HMQ  hmq;
    QMSG qmsg;
    HAB  hab;

    ULONG flFrameFlags=
        FCF_TITLEBAR      | /* Title bar          */
        FCF_SIZEBORDER    | /* Size Border        */
        FCF_MINMAX        | /* Min & Max Buttons  */
        FCF_SYSMENU       | /* System Menu        */
        FCF_SHELLPOSITION | /* System size & position */
        FCF_TASKLIST      | /* Add name to task list */
        FCF_ICON          | /* Add icon           */
        FCF_ACCELTABLE     | /* Add accelerator table */
        FCF_MENU;         | /* Add menu           */

    hab = WinInitialize(0);
    hmq = WinCreateMsgQueue(hab, 0);
    WinRegisterClass(
        hab,                /* Anchor block handle */
        "ResSamClient",     /* Name of class being registered */
        (PFNWP)ClientWndProc, /* Window procedure for class */
        CS_SIZEREDRAW |     /* Class style          */
        CS_HITTEST,        /* Class style          */
        0);                /* Extra bytes to reserve */

    hwndFrame = WinCreateStdWindow(
        HWND_DESKTOP,      /* Parent is desktop window */
        WS_VISIBLE,        /* Make frame window visible */
        &flFrameFlags,      /* Frame controls            */
        "ResSamClient",    /* Window class for client   */
        NULL,              /* No window title           */
        WS_VISIBLE,        /* Make client window visible */
        (HMODULE) 0,        /* Resources in application module */
        ID_RESOURCE,       /* Resource identifier        */
        NULL);             /* Pointer to client window handle */

    while (WinGetMsg(hab, &qmsg, 0, 0, 0))
        WinDispatchMsg(hab, &qmsg);
    WinDestroyWindow(hwndFrame);
}

```

```

WinDestroyMsgQueue(hmq);
WinTerminate(hab);
return 0;
}
MRESULT EXPENTRY ClientWndProc(HWND hwnd,ULONG msg,MPARAM mp1,MPARAM mp2);
{
RECTL rcl;
HPS hps;
static LONG lColor=CLR_RED;
switch (msg)
{
case WM_PAINT:
hps=WinBeginPaint(hwnd,(HPS) NULL, &rcl); /* Get hps */
WinFillRect(hps,&rcl,lColor); /* Fill the window */
WinEndPaint(hps); /* Free hps */
return 0;
case WM_COMMAND:
switch (SHORT1FROMMP(mp1))
{
case IDM_SHIFT: /* Shift selected */
if (lColor==CLR_RED) /* Change the color */
lColor=CLR_BLUE;
else lColor=CLR_RED;
WinInvalidateRect(hwnd,(PRECTL)NULL,0UL); /* Paint Window */
return 0;
case IDM_EXIT: /* Exit selected */
WinPostMsg(hwnd,WM_CLOSE,MPVOID,MPVOID); /* Exit program */
return 0;
}
}
return WinDefWindowProc (hwnd, msg, mp1, mp2);
}

```

Frame-Window Class Data

An application can specify class-specific data for a frame window by passing to WinCreateWindow a pointer to the FRAMECDATA structure. The class-specific data contains the frame-control flags (FCF_ flags), resource-module handle, and resource identifier to be used when creating the frame window. The resource-module handle and the resource identifier specify where to find resources for the frame window.

Supplying class-specific data with WinCreateWindow is similar to using WinCreateStdWindow without creating a client window.

Frame-Window Data

Frame-window data specifies the state of the frame window at a given time. An application can retrieve the frame-window data by calling WinQueryWindowUShort. A frame window has the following state flags:

Flag	Description
FF_ACTIVE	Indicates that the frame window is active.
FF_DLGDISMISSED	Indicates that a dialog window has been dismissed by a call to WinDismissDlg. If the dialog window needs to be processed through WinProcessDlg , after being dismissed, the FF_DLGDISMISSED flag must be reset.
FF_FLASHHILITE	Indicates that the frame window is

	flashing and its flash state is TRUE.
FF_FLASHWINDOW	Indicates that the frame window flashes as the result of either a call to WinFlashWindow or a WM_FLASHWINDOW message.
FF_NOACTIVATESWP	Indicates that the system should do no z-ordering on this frame window.
FF_OWNERDISABLE	Indicates whether the owner window was enabled or disabled when the dialog window was loaded, for a frame window that is part of a dialog window,
FF_OWNERHIDDEN	Indicates that the frame window's owner window is hidden or minimized, in which case the frame window also is hidden.
FF_SELECTED	Indicates that the frame window has been selected.
FI_ACTIVATEOK	Indicates that the window can be activated.
FI_FRAME	Indicates that the window is a frame window.
FI_NOMOVEWITHOWNER	Indicates that the window should move when its owner window moves.
FI_OWNERHIDE	Indicates that the frame window should be hidden or shown as a result of its owner window being hidden, shown, minimized, or maximized.

Frame-Window Operation

The frame window maintains the size, position, and visibility of itself, its frame controls, and its client window. The frame window responds to user requests to move, size, minimize, maximize, and redraw itself. It also responds to requests to close (destroy) itself and to change the focus and activation state.

The frame window, when being moved or sized, maintains the position of each owned window relative to its owner window's lower-left corner.

Whenever the frame window redraws itself (for example, after being moved or sized), it draws the frame controls and then lets the application draw the client window. This order ensures that the rapidly drawn frame controls are drawn before the client window.

The order in which the frame controls are drawn depends on the z-order position of the controls. The following list specifies the z-order position of the frame controls (from top to bottom):

```
FID_SYSMENU
FID_TITLEBAR
FID_MENU
FID_VERTSCROLL
FID_HORZSCROLL
FID_CLIENT
```

Although an application can change the z-order position of any window, changing the relative positions of frame controls is not recommended.

When the user maximizes the frame window, the size of the frame window increases to the size of its parent window, plus an additional amount on each of its four sides equal to the width of its sizing border. A window always is clipped to its parent window; a maximized standard frame window does not show its sizing border in its normal maximized position.

Frame controls owned by a frame window or windows owned by child windows of a frame window are destroyed automatically when the

frame window processes the WM_DESTROY message.

Nonstandard Frame Windows

Although most applications use frame windows to create their main windows and dialog windows, they are not limited to frame windows. Applications can create nonstandard frame windows and still use the standard frame controls, such as the title bar and system menu, within the nonstandard windows.

An application can create a nonstandard frame window either by subclassing a frame window or by creating a private frame-window class. An application that subclasses a frame window can intercept the messages sent to the window and process them in new ways. An application that creates private frame-window classes essentially rewrites the frame-window procedure. In either case, by creating nonstandard frame windows, the application gains much more control over the arrangement of frame controls in the frame window.

The messages WM_FORMATFRAME, WM_UPDATEFRAME, and WM_CALCVALIDRECTS control the arrangement of frame controls for applications that subclass the frame-window procedure. By intercepting these messages, an application can rearrange the frame controls in a frame window.

To maintain the size and position of frame controls, an application that creates private frame-window classes can use WinCreateFrameControls and WinCalcFrameRect. These functions provide capabilities that are similar to those provided by frame windows.

Default Frame-Window Behavior

The following table lists all the messages specifically handled by the window procedure of the predefined frame-window class (WC_FRAME) and describes how the window procedure responds to each message.

Message	Description
WM_ACTIVATE	Sets the highlighted state of the title bar or border so that it matches the frame window's activation state.
WM_BUTTON1DOWN	If the frame window is minimized, captures the mouse; otherwise, activates the frame window.
WM_BUTTON2DOWN	Activates the frame window.
WM_BUTTON3DOWN	Activates the frame window.
WM_BUTTON1UP	Processes messages from minimized window frames.
WM_BUTTON1DBLCLK	If the frame window is minimized, posts a WM_SYSCOMMAND message to itself; otherwise, activates the frame window.
WM_CALCVALIDRECTS	If the frame window has no client window or if the client window has the CS_SIZEREDRAW style, returns the CVR_REDRAW flag to invalidate the entire window.
WM_CLOSE	If the frame window has a client window, passes this message to the client; otherwise, returns the result of WinDefWindowProc.
WM_CREATE	Creates the specified frame controls by calling WinCreateFrameControls. Also creates any accelerator tables, loads icons, and adds itself to the Window List. These actions depend on the frame-window styles and frame-control flags specified for the window.

<code>WM_DESTROY</code>	If the focus is held by a child window of the frame window, sets the focus to the frame window's parent window, destroys any owned windows or child windows, destroys any icons created by using the <code>FS_ICON</code> style, and destroys any accelerator tables created by using the <code>FS_ACCELTABLE</code> style.
<code>WM_ENABLE</code>	Returns the result of <code>WinDefWindowProc</code> .
<code>WM_ERASEBACKGROUND</code>	Returns <code>TRUE</code> , signaling that the window should erase the client-window area. The frame window sends this message to itself during <code>WM_PAINT</code> processing.
<code>WM_FORMATFRAME</code>	Calculates the sizes and positions of the frame controls and the client window.
<code>WM_HITTEST</code>	If the frame window is minimized and disabled, returns <code>HT_ERROR</code> ; otherwise, returns <code>TF_MOVE</code> .
<code>WM_MINMAXFRAME</code>	If the frame window has a client window, passes this message to the client window; otherwise, passes this message to <code>WinDefWindowProc</code> .
<code>WM_MOUSEMOVE</code>	Determines the correct mouse pointer to use and returns the result of <code>WinDefWindowProc</code> .
<code>WM_PAINT</code>	If the frame window is minimized, sends <code>WM_QUERYICON</code> and <code>WM_ERASEBACKGROUND</code> to itself and draws the icon; otherwise, paints the control windows, sends a <code>WM_ERASEBACKGROUND</code> message to the client window, and paints the client window.
<code>WM_QUERYTRACKINFO</code>	Starts track-move processing of the title-bar control window.
<code>WM_SHOW</code>	Returns the result of <code>WinDefWindowProc</code> .
<code>WM_SIZE</code>	Sends a <code>WM_FORMATFRAME</code> message to itself.
<code>WM_SYSCOMMAND</code>	If the frame window has captured the mouse, ignores the system command; otherwise, uses one of the following commands: <code>SC_APPMENU</code> , <code>SC_CLOSE</code> , <code>SC_MOVE</code> , <code>SC_NEXT</code> , <code>SC_NEXTFRAME</code> , <code>SC_RESTORE</code> , <code>SC_SIZE</code> , <code>SC_SYSMENU</code> , <code>SC_TASKMANAGER</code> .
<code>WM_UPDATEFRAME</code>	Reformats and updates the appearance of the frame window. Sent after a frame control has been added to or removed from the frame window.

Using Frame Windows

This section explains how to:

- Create a main window
- Retrieve a frame-control handle

Creating a Main Window

An application can create a main window by using `WinCreateStdWindow`. The following code fragment creates a typical main window—a frame window that has a system menu, title bar, menu, vertical and horizontal scroll bars, minimize and maximize (window-sizing) buttons, and a sizing border:

```
#define IDM_MENU 1

HWND hwndFrame;
ULONG flFrameControlFlags =
    FCF_SYSMENU      |
    FCF_TITLEBAR     |
    FCF_SIZEBORDER   |
    FCF_MENU         |
    FCF_MINMAX       |
    FCF_HORZSCROLL   |
    FCF_VERTSCROLL   |
    FCF_SHELLPOSITION;

hwndFrame = WinCreateStdWindow(
    HWND_DESKTOP,          /* Frame-window parent */
    WS_VISIBLE,            /* Make window visible */
    &flFrameControlFlags,  /* Frame-control flags */
    "MyClass",             /* Client-window class */
    "Main Window",        /* Window title */
    0,                    /* No client-window styles */
    (HMODULE) NULL,       /* App. module has resources */
    IDM_MENU,             /* Resource ID */
    0);                   /* Client-window handle */
```

An application also can create a *standard* main window by creating a frame window with the `FCF_STANDARD` flag. The application must include icon, menu, and accelerator-table resources if it uses the `FCF_STANDARD` flag.

The application creates the standard window by using `WinCreateStdWindow`, as shown in the following code fragment:

```
#define IDM_RESOURCES 1

HWND hwndFrame;

/* Set the frame-control flags. */
ULONG flFrameControlFlags = FCF_STANDARD;

/* Create the standard main window. */
hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
    WS_VISIBLE,
    &flFrameControlFlags,
    "MyClass",
    "Main Window", 0,
    (HMODULE) NULL,
    IDM_RESOURCES, 0);
```

Another way to create a main window and its frame controls is to use `WinCreateWindow` to create the frame window and the frame controls, then call `WinCreateWindow` again to create the client window. One advantage of this approach is that, when creating the frame window, the application can specify the window's initial size and position. The following figure illustrates this approach:

```
#define ID_RESOURCES 1
#define ID_FRAME 1
ULONG flFrameControlFlags =
    FCF_ACCELTABLE |
    FCF_ICON       |
    FCF_MENU       |
    FCF_MINMAX     |
    FCF_SIZEBORDER |
    FCF_SYSMENU    |
```

```

FCF_TASKLIST |
FCF_TITLEBAR;

FRAMECDATA fcdata;
HWND hwndFrame;
HWND hwndClient;
SWP swp;

fcdata.cb          = sizeof(FRAMECDATA);
fcdata.flCreateFlags = flFrameControlFlags;
fcdata.hmodResources = (HMODULE) NULL;
fcdata.idResources  = ID_RESOURCES;

/* Create the frame and client windows. */
hwndFrame = WinCreateWindow(
    HWND_DESKTOP,    /* Frame-window parent */
    WC_FRAME,        /* Frame-window class */
    "Main Window",   /* Window title */
    0,               /* Initially invisible */
    0,0,0,0,         /* Size and position = 0 */
    NULL,            /* No owner */
    HWND_TOP,        /* Top z-order position */
    ID_FRAME,        /* Frame-window ID */
    &fcdata,         /* Pointer to class data */
    NULL);           /* No presentation parameters */

hwndClient = WinCreateWindow(
    hwndFrame,       /* Client-window parent */
    "MyClass",       /* Client-window class */
    NULL,            /* No title for client window */
    0,               /* Initially invisible */
    0,0,0,0,         /* Size and position = 0 */
    hwndFrame,       /* Owner is frame window */
    HWND_BOTTOM,     /* Bottom z-order position */
    FID_CLIENT,      /* Standard client-window ID */
    NULL,            /* No class data */
    NULL);           /* No presentation parameters */

.
. /* Continue with initialization. */
.

/* Set the size and position of the frame window. */
WinQueryWindowPos(HWND_DESKTOP, &swp);
WinSetWindowPos(hwndFrame,
    HWND_TOP,
    swp.x,
    swp.cy / 2,
    swp.cx,
    swp.cy / 2,
    SWP_MOVE |
    SWP_SIZE);

/* Set the size and position of the client window. */
WinQueryWindowPos(hwndFrame, &swp);
WinSetWindowPos(hwndClient,
    HWND_TOP,
    SV_CXSIZEBORDER,
    SV_CYSIZEBORDER - 1,
    swp.cx - SV_CXSIZEBORDER * 2,
    (swp.cy - SV_CYSIZEBORDER * 2) + 1,
    SWP_MOVE |
    SWP_SIZE);

/* Make the frame and client windows visible. */
WinShowWindow(hwndFrame, TRUE);
WinShowWindow(hwndClient, TRUE);

```

Retrieving a Frame Handle

An application can retrieve a frame-control handle by using WinWindowFromID. The following code fragment retrieves the handle of a title-bar control:

```
HWND hwndTitleBar,hwndFrame;  
hwndTitleBar = WinWindowFromID(hwndFrame, FID_TITLEBAR);
```

Given a frame-control handle, an application can retrieve its parent frame-window handle by using WinQueryWindow:

```
HWND hwndFrame,hwndTitleBar;  
hwndFrame = WinQueryWindow(hwndTitleBar, QW_PARENT);
```

By using identifiers to identify frame controls, rather than using window classes, an application can create its own controls to replace the predefined controls.

Hooks

A *hook* is a point in a system-defined function where an application can supply additional code that the system processes as though it were part of the function. This chapter describes how to use hooks in PM applications.

About Hooks

Many operating system functions provide points where an application can *hook in* its own code to enhance or override the default processing of the function. Most hooks enable an application to monitor some aspect of the message stream. For example, the input hook enables an application to monitor all messages posted to a particular message queue.

A hook function can be associated with the system-message queue, so that it monitors messages for all applications. These system-queue hook functions can be called in the context of any application. However, they must be defined in separate dynamic link library (DLL) modules, because it is not possible to call application-module procedures from other applications.

A hook function can also be associated with the message queue of an individual thread, so that it monitors messages for that thread only. These message-queue hook functions are called only in the context of the thread. Therefore, these hook functions are typically defined locally.

OS/2 operating system contains many types of hooks, and the system maintains a separate *hook list* for each type of hook supported.

Hook Lists

A *hook list* contains the addresses of the functions that the system calls while processing a hook. An application can take advantage of a particular type of hook by defining a hook function and using WinSetHook to enter the address of the function in the corresponding hook list. To specify the hook type in WinSetHook, the application uses one of the following constants:

Constant Name	Description
HK_CHECKMSGFILTER	Lets applications apply very specific message filtering. See HK_CHECKMSGFILTER - Check Message Filter Hook .

HK_CODEPAGECHANGED	Lets applications determine when the code page changes. See HK_CODEPAGECHANGE - Code Page Changed Hook .
HK_DESTROYWINDOW	Called whenever a window is destroyed. See HK_DESTROYWINDOW - Destroy Window Hook .
HK_FINDWORD	Lets applications control where WinDrawText places line breaks. See HK_FINDWORD - Find Word Hook .
HK_FLUSHBUF	Lets applications save data before the system reboots. See HK_FLUSHBUF - Flush Buffer Hook .
HK_HELP	Monitors the WM_HELP message. See HK_HELP - Help Hook .
HK_INPUT	Monitors messages in the specified message queue. See HK_INPUT - Input Hook .
HK_JOURNALPLAYBACK	Lets applications insert messages into the system message queue. See HK_JOURNALPLAYBACK - Journal Playback Hook .
HK_JOURNALRECORD	Lets applications record mouse and keyboard input messages. See HK_JOURNALRECORD - Journal Record Hook .
HK_LOADER	Lets the library and procedure loading and deleting calls be intercepted. See HK_LOADER - Loader Hook .
HK_LOCKUP	Called when the system locks itself up. See HK_LOCKUP - Lockup Hook .
HK_MSGCONTROL	Monitors the flow of messages to be intercepted. See HK_MSGCONTROL - Message Control Hook .
HK_MSGFILTER	Monitors input events during system modal loops. See HK_MSGFILTER - Message Filter Hook .
HK_MSGINPUT	Lets applications simulate user input, and only mouse and keyboard messages should be passed in. All other messages will be discarded. Mouse and keyboard messages injected into this hook will have the same effect as if they were generated by the mouse or keyboard device driver. The messages are routed in the same manner as normal user input. See HK_MSGINPUT - Message Input Hook .
HK_PLIST_ENTRY	Called every time a program-list call or initialization file call is invoked by an application. It is called before the call is run. See HK_PLIST_ENTRY - Program List Call Hook .
HK_PLIST_EXIT	Called every time a program-list call or initialization file call is invoked by an application. It is called before the call is run. See HK_PLIST_EXIT - Program List Exit Hook .
HK_REGISTERUSERMSG	Called whenever a user message or data type is registered. See HK_REGISTERUSERMSG - Register User Message Hook .
HK_SENDMSG	Monitors messages sent by using WinSendMsg. See HK_SENDMSG - Send Message Hook .

HK_WINDOWDC	Called when a device context is allocated or freed. See HK_WINDOWDC - Device Context Hook .
-------------	---

While running a function that contains a hook, the system checks for any function addresses in the hook list that correspond to the type of hook. If an address is found, the system tries to locate and run the function.

Hook Chains

In the hook lists associated with most message-monitoring hooks, the function addresses are linked to form chains. The system passes a message to each hook function in the list, one after the other. Each function can modify the message or stop its progress through the chain, thereby preventing it from reaching the next hook or the destination window. The system calls chained hook functions in last-installed, first-called order.

Hook Types

Each type of hook passes a characteristic set of arguments to the functions referenced in the corresponding hook list. For an application to use a particular hook, it must define a function that processes those arguments and enter the address of the function in the hook list using WinSetHook. This section describes the types of hooks available in OS/2 operating system and the requirements of the functions that process each hook type.

HK_CHECKMSGFILTER - Check Message Filter Hook

The *check message filter hook* is called whenever WinGetMsg, WinWaitMsg, or WinPeekMsg are used to filter message identities. This hook lets an application apply very specific message filtering, for example, based on the values of message parameters. This hook is called after window handle filtering and before message filtering. The following code shows the syntax for a check message filter hook function:

```
BOOL WINAPI CheckMsgFilterHook ( HAB    hab,
                                PQMSG  pQmsg,
                                ULONG   usFirst,
                                ULONG   usLast,
                                ULONG   fOptions);
```

The *hab* parameter is the anchor block handle.

The *pQmsg* parameter is a pointer to a QMSG data structure that contains information about the message.

The *usFirst* parameter is the first message identity specified on a call to the WinGetMsg, WinPeekMsg, or WinWaitMsg function.

The *usLast* parameter is the last message identity specified on a call to the WinGetMsg, WinPeekMsg, or WinWaitMsg function.

The *fOptions* parameter indicates whether or not the message is removed from the queue:

```
PM_NOREMOVE
PM_REMOVE
```

If the check message filter hook function returns TRUE, the message is accepted by the filtering. Any further check message filter hooks in the chain are ignored, any filtering specified by the WinGetMsg, WinPeekMsg, and WinWaitMsg functions are ignored, and processing of the message continues.

A hook that always returns TRUE effectively switches off message filtering.

If the check message filter hook function returns FALSE, the message is passed on to the next check message filter hook in the chain. If the end of the chain has been reached, the filtering specified by the WinGetMsg, WinPeekMsg, or WinWaitMsg functions is applied.

HK_CODEPAGECHANGE - Code Page Changed Hook

The *code page changed hook* notifies an application when the code page associated with the specified message queue has been changed. The system calls a code page changed hook function after setting the new code page. Typically, the code page changed hook is used in applications that support multiple languages. The following code shows the syntax for a code page changed hook function:

```
VOID EXPENTRY CodePageChangedHook(HMQ hmq,
                                   USHORT usOldCodepage,
                                   USHORT usNewCodepage);
```

The *hmq* parameter receives the handle of the message queue that is changing its code page. The *usOldCodepage* is the code page identifier of the previous code page.

The *usNewCodepage* parameter is the identifier of the new code page.

A code page changed hook function does not return a value, and the system always calls the next function in the chain.

HK_DESTROYWINDOW - Destroy Window Hook

The *destroy window hook* is called whenever a window is destroyed. The following code shows the syntax for a destroy window hook function:

```
BOOL EXPENTRY DestroyWindowHook (HAB hab,
                                  HWND hwnd,
                                  ULONG ulReserved);
```

This hook is sent after the WM_DESTROY message has been sent and just before the window becomes invalid.

The *hab* parameter is the anchor block handle.

The *hwnd* parameter is the handle of the window being destroyed.

The *ulReserved* parameter is reserved.

When this hook function returns TRUE, the function completed successfully.

When it returns FALSE, an error occurred.

HK_FINDWORD - Find Word Hook

The *find word hook* allows an application to control where WinDrawText breaks a character string that is too wide for the drawing rectangle. If the DT_WORDBREAK flag is set, the system calls this hook from within WinDrawText. Typically, this hook is used to avoid awkward line breaks in applications that use double-byte character sets. The following code shows the syntax for a find word hook function:

```
BOOL EXPENTRY FindWordHook(USHORT usCodepage,
                           PSZ pszText,
                           ULONG cb,
                           ULONG ich,
                           PULONG pichStart,
                           PULONG pichEnd,
```

```
PULONG pichNext);
```

The *usCodePage* parameter contains the code page identifier of the string to be formatted; the *pszText* parameter contains a pointer to the actual string.

The *cb* parameter contains a value specifying the number of bytes in the string. This value is 0 if the string is null-terminated.

The *ich* parameter contains the index of the character in the string that intersects the right edge of the drawing rectangle.

A find word hook function uses these four parameters to determine the word that contains the intersecting character. It then fills the remaining three parameters, *pichStart*, *pichEnd*, and *pichNext*, with the indexes of the starting character of the word, ending character of the word, and starting character of the next word in the string.

If the find word hook function returns TRUE, WinDrawText draws the string only up to, but not including, the specified word. If the function returns FALSE, WinDrawText formats the string in the default manner.

HK_FLUSHBUF - Flush Buffer Hook

The *flush buffer hook* allows applications to save data before the system reboots. The following code shows the syntax for a flush buffer hook function:

```
BOOL WINAPI FlushBufHook (HAB hab);
```

This hook is called to notify applications that the system is rebooting due to a Ctrl+Alt+Del sequence being entered. It enables applications to write existing information to the hardfile immediately, avoiding loss of data before the system reboots.

Note: Do not use any of the Win or Gpi functions inside the hook routine. At the point in time that the hook routine is running, these subsystems might not be fully available because they might be partially shutdown.

The *hab* parameter is the application anchor block.

When this function returns TRUE, the function completed successfully.

When this function returns FALSE, an error occurred.

HK_HELP - Help Hook

The *help hook* allows an application to include online help. The system calls a help hook function during the default processing of the WM_HELP message. Help processing is done in two stages: creating the WM_HELP message and calling the help hook. The WM_HELP message can come from the following sources:

- WM_CHAR message, after translation by an ACCEL data structure with the AF_HELP style. The default system accelerator table translates the F1 key into a help message. The WM_HELP message is posted to the current focus window, which can be a menu, a button, a frame, or your client window.
- Menu-bar selection, when the MIS_HELP style is specified for the menu-bar item. The WM_HELP message is posted to the current focus window.
- Dialog-window push button, when the BS_HELP style is specified for the push button. The WM_HELP message is posted to the owner window of the button, which normally is the dialog window.
- Message box, when the MB_HELP style is specified for the message box. The WM_HELP message is posted to the message box.

The WM_HELP message is posted to the current focus window. The default processing in WinDefWindowProc is to pass the message up to the parent window. If the message reaches the client window, it can be processed there. If the message reaches a frame window, the default frame-window procedure calls the help hook. The help hook is also called if a WM_HELP message is generated while the application is in menu mode, that is, while a selection is being made from a menu. The following code shows the syntax for a help hook function:

```

BOOL EXPENTRY HelpHook(HAB hab, ULONG usMode, ULONG idTopic,
                      ULONG idSubTopic, PRECTL prcPosition)

```

If a help hook function returns TRUE, the system does not call the next help hook function in the chain. If the function returns FALSE, the system calls the next help hook function in the chain. The arguments passed to the function provide contextual information, such as the screen coordinates of the focus window and whether the message originated in a message box or a menu.

The WM_HELP message often goes to a frame window instead of to the client window. The frame window processes a WM_HELP message as follows:

- If the window with the focus is the FID_CLIENT window, the frame window passes the WM_HELP message to the FID_CLIENT window.
- If the parent of the window with the focus is the FID_CLIENT frame-control window, the frame window calls the help hook, specifying the following:

```

Mode      = HLPM_FRAME
Topic     = frame-window identifier
Subtopic  = focus-window identifier
Position  = screen coordinates of focus window

```

- If the parent of the focus window is not an FID_CLIENT window (it could be the frame window or a second-level dialog window), the frame window calls the help hook, specifying the following:

```

Mode      = HLPM_WINDOW
Topic     = identifier of parent of focus window
Subtopic  = focus-window identifier
Position  = screen coordinates of focus window

```

An application receives the WM_HELP message in its dialog-window procedure. The application can ignore the message, in which case the frame-window action occurs as described, or the application can handle the WM_HELP message directly.

Menu windows receive a WM_HELP message when the user presses the Help accelerator key (F1 by default) while a menu is displayed. Menu windows process WM_HELP messages by calling the help hook, specifying the following:

```

Mode      = HLPM_MENU
Topic     = identifier of pull-down menu
Subtopic  = identifier of selected item in pull-down menu
Position  = screen coordinates of selected item

```

A help hook function should respond by displaying information about the selected menu item.

WinDefWindowProc processes WM_HELP messages by passing the message to the parent window. Typically, the message moves up the parent chain until it arrives at a frame window.

HK_INPUT - Input Hook

The *input hook* enables an application to monitor the system-message queue or an application-message queue. The system calls an input-hook function whenever WinGetMsg or WinPeekMsg is about to return a message. Typically, an application uses the input hook to monitor mouse and keyboard input and other messages posted to a queue. The following code shows the syntax for an input-hook function:

```

BOOL EXPENTRY InputHook(HAB hab, PQMSG pQmsg, ULONG fs)

```


The *pQmsg* parameter is a pointer to a QMSG data structure that contains information about the message.

The */s* parameter of InputHook can contain the following flags from WinPeekMsg, indicating whether or not the message is removed from the queue:

```
PM_NOREMOVE  
PM_REMOVE
```

If an input-hook function returns TRUE, the system does not pass the message to the rest of the hook chain or to the application. If the function returns FALSE, the system passes the message to the next hook in the chain or to the application if no other hooks exist.

An input-hook function can modify a message by changing the contents of the QMSG data structure, then returning FALSE to pass the modified message to the rest of the chain. The following problems can occur when a hook modifies a message:

- If the caller uses WinPeekMsg or WinGetMsg with a message filter range (msgFilterFirst through msgFilterLast), the message is checked before the hook functions are called, not after. If the input-hook function modifies the *msg* field of the QMSG data structure, the caller can receive messages that are not in the range of the message filter of the caller.
- If the input-hook function changes a WM_CHAR message from one character into another—for example, if the function modifies all Tab messages into F6 messages—an application that depends on the key state is unable to interpret the result. (When the Tab key is translated into the F6 key, the application receives the F6 keystroke and enters a process loop, waiting for the F6 key to be released; the application calls WinGetKeyState with the HWND_DESKTOP and VK_F6 arguments).

HK_JOURNALPLAYBACK - Journal Playback Hook

The *journal playback hook* enables an application to insert messages into the system-message queue. Typically, an application uses this hook to play back a series of mouse and keyboard events that were recorded earlier using the journal record hook. A journal playback hook function can be associated only with the system-message queue.

Regular mouse and keyboard input is disabled as long as a journal playback hook is installed. It is important to notice that, because mouse and keyboard input are disabled, this hook can easily hang the system. The following code shows the syntax for a journal playback hook function:

```
ULONG WINAPI JournalPlaybackHook(HAB hab, BOOL fSkip,  
                                PQMSG pQmsg)
```

The *pQmsg* parameter is a pointer to a QMSG data structure that the journal playback hook function fills in with the message to be played back. If the *fSkip* parameter is FALSE, the function fills in the QMSG data structure with the current recorded message. The function returns the same message each time it is called, until *fSkip* is TRUE. The same message is returned many times if an application is examining the queue but not removing the message. If *fSkip* is TRUE, the function advances to the next message without filling in the QMSG data structure, because the *pQmsg* parameter is NULL when *fSkip* is TRUE.

The journal playback hook returns a ULONG time-out value that tells the system how many milliseconds to wait before processing the current message from the playback hook. This enables the hook to control the timing of the events it plays back.

The *time* field of the QMSG data structure is filled in with the current time before the playback hook is called. The hook should use the time stored in this field, instead of the system clock, to set up delays between events.

HK_JOURNALRECORD - Journal Record Hook

The *journal record hook* allows an application to monitor the system-message queue and to record input events. Typically, an application uses this hook to record a sequence of mouse and keyboard events that it can play back later by using the journal playback hook. A journal record hook function can be associated only with the system-message queue. The following code shows the syntax for a journal record hook function:

```
VOID WINAPI JournalRecordHook(HAB hab, PQMSG pQmsg)
```

The *pQmsg* parameter is a pointer to a QMSG data structure containing information about the message. The system calls the journal record hook function after processing the raw input enough to create valid WM_CHAR or mouse messages and after setting the *window-handle* field of the QMSG data structure.

A journal record hook function does not return a value, and the system always calls the next function in the chain. Typically, a journal record hook function saves the input events to a disk file to be played back later. The *hwnd* field of the QMSG data structure is not important and is ignored when the message is played back.

The following messages are passed to the journal record hook:

```
WM_CHAR
WM_BUTTON1DOWN
WM_BUTTON1UP
WM_BUTTON2DOWN
WM_BUTTON2UP
WM_BUTTON3DOWN
WM_BUTTON3UP
WM_MOUSEMOVE.
```

The positions stored in the mouse messages are in screen coordinates. The system does not combine mouse clicks into double clicks before calling the hook, because there is no guarantee that both clicks will be in the same window when they are played back.

The system passes a WM_JOURNALNOTIFY message to the journal record hook function whenever an application calls WinGetPhysKeyState or WinQueryQueueStatus. This message is necessary because the system-message queue is only one message deep while a playback hook is active. For example, the user might press the A, B, and C keys while in record mode. While the application is processing the *A* character message, the B key might be down; WinGetPhysKeyState returns this information. However, during playback mode, the system knows only that it currently is processing the A key.

HK_LOADER - Loader Hook

The *loader hook* allows the library and procedure loading and deleting calls to be intercepted. The following code shows the syntax for a loader hook function:

```
BOOL WINAPI LoaderHook(HAB hab,
                      LONG idContext,
                      PSZ pszLibname,
                      PHLIB hlib,
                      PSZ pszProcname,
                      PFNWP wndProc);
```

If the hook attempts a load or deletion which is unsuccessful, then the hook must establish the relevant error information.

The *hab* parameter is the anchor block handle.

The *idContext* parameter is the origin of the call to the hook:

```
LHK_DELETEPROC WinDeleteProcedure
LHK_DELETELIB WinDeleteLibrary
LHK_LOADPROC WinLoadProcedure
LHK_LOADLIB WinLoadLibrary
```

The *pszLibname* parameter is the library name.

The *hlib* parameter is a pointer to a library handle.

If the *idContext* parameter is set to LHK_LOADLIB, then this hook must set the value of this parameter to the handle of the loaded library or to NULLHANDLE if the load fails.

The *pszProcname* parameter is the procedure name.

The *wndProc* parameter is the window procedure identifier.

If the *idContext* parameter is set LHK_LOADPROC, then this hook must set the value of this parameter to the handle of the loaded procedure or to NULL if the load fails.

The *pfSuccess* parameter is the success indicator, which is either TRUE or FALSE. If it is TRUE, the library or procedure loaded or deleted successfully. If it is FALSE, the library or procedure not loaded or deleted successfully.

When this function returns TRUE, it does not call the next hook in chain. When this function returns FALSE, then it does call the next hook in chain.

HK_LOCKUP - Lockup Hook

The *lockup hook* is called when the system locks itself up. The following code shows the syntax for a lockup hook function:

```
BOOL WINAPI LockupHook (HAB hab,
                       HWND hwndLockupFrame);
```

The *hab* parameter is the application anchor block.

The *hwndLockupFrame* parameter is the frame window of the lockup panel.

This function has no return value.

All HK_LOCKUP hooks registered with the system are called when the system locks itself up. All HK_LOCKUP hooks must be system hooks, not message queue hooks.

Application programs that create other lockup password input windows by hooking the HK_LOCKUP system hook can use WinUnlockSystem to force the system to unlock when another form of input is detected other than mouse or keyboard. For example, a pen gesture or some voice input or signature recognition.

HK_MSGCONTROL - Message Control Hook

The *message control hook* allows the application to determine the flow of messages to be intercepted. The following code shows the syntax for a message control hook function:

```
BOOL WINAPI MsgControlHook(HAB hab,
                           LONG/SHORT idContext,
                           HWND hwnd,
                           PSZ pszClassname,
                           ULONG/USHORT usMsgclass,
                           LONG/SHORT idControl,
                           PBOOL fSuccess);
```

If the hook is unable to alter the message control state, then the hook must establish the relevant error information.

The *hab* parameter is the anchor block handle.

The *idContext* parameter is the origin of the call to the hook and has one of the following values:

MCHK_CLASSMSGINTEREST
WinSetClassMsgInterest

MCHK_MSGINTEREST
WinSetMsgInterest

MCHK_MSGMODE WinSetMsgMode

MCHK_SYNCHRONISATION
WinSetSynchroMode

The *hwnd* parameter is the window handle.

The *pszClassName* parameter is the window class name.

The *usMsgClass* parameter is the message class.

The *idControl* parameter is the control setting.

The *lSuccess* parameter is the success indicator and is either TRUE (success) or FALSE (error).

This function returns either TRUE or FALSE. If it returns TRUE, the next hook in the chain is not called. If it returns FALSE, the next hook in the chain is called.

HK_MSGFILTER - Message Filter Hook

The *message-filter hook* allows an application to provide input filtering (such as monitoring hot keys) during system-modal loops. The system calls a message-filter hook function while tracking the window size and movement, displaying a modal dialog window or message box, tracking a scroll bar, and during window-enumeration operations. The following code shows the syntax for a message-filter hook function:

```
BOOL WINAPI MsgFilterHook(HAB hab, PQMSG pQmsg, ULONG msgf)
```

The *msgf* parameter can have one of the three values shown in the following table:

Parameter Value	Description
MSGF_DIALOGBOX	Message originated while processing a modal dialog window or a message box.
MSGF_MESSAGEBOX	Message originated while processing a message box.
MSGF_TRACK	Message originated while tracking a control (such as a scroll bar).

The *pQmsg* parameter of `MsgFilterHook` is a pointer to a `QMSG` data structure containing information about the message.

If a message-filter hook function returns TRUE, the system does not pass the message to the rest of the hook chain or to the application. If the function returns FALSE, the system passes the message to the next hook function in the chain or to the application if no other functions exist.

This hook enables applications to perform message filtering during modal loops that is equivalent to the typical filtering for the main message loop. For example, applications often examine a new message in the main event loop between the time they retrieve the message from the queue and the time they dispatch it, performing special processing as appropriate. An application usually cannot do this sort of filtering during a modal loop, because the system runs the loop created by `WinGetMsg` and `WinDispatchMsg`. If an application installs a message-filter hook function, the system calls the function between `WinGetMsg` and `WinDispatchMsg` in the modal processing loop.

An application can also call the message-filter hook function directly by calling `WinCallMsgFilter`. With this function, the application can use the same code as the main message loop to filter messages during modal loops. To do so, the application encapsulates the filtering operations in a message-filter hook function and calls `WinCallMsgFilter` between `WinGetMsg` and `WinDispatchMsg` calls, as shown in the following code fragment:

```
while (WinGetMsg(hab, (PQMSG) &qmsg, (HWND) NULL, 0, 0))
{
    if (!WinCallMsgFilter(hab, (PQMSG) &qmsg, 0))
        WinDispatchMsg(hab, (PQMSG) &qmsg);
}
```

The last argument of WinCallMsgFilter is passed to the hook function; the application can enter any value. By defining a constant such as MSGF_MAINLOOP, the hook function can use that value to determine from where the function was called.

HK_MSGINPUT - Message Input Hook

The *message input hook* is intended for simulated user input, and only mouse and keyboard messages should be passed in. All other messages will be discarded. Mouse and keyboard messages injected into this hook will have the same effect as if they were generated by the mouse or keyboard device driver. The messages are routed in the same manner as normal user input. The following code shows the syntax for a message input hook function:

```
BOOL WINAPI MsgInputHook (HAB hab,
                          PQMSG pQmsg,
                          BOOL fSkip,
                          PBOOL pfNoRecord);
```

The *hab* parameter is the anchor block handle.

The *pQmsg* parameter is the queue message data structure to be filled in with a simulated mouse or keyboard message.

The *fSkip* parameter is the skip message flag, which is either TRUE or FALSE. If TRUE, the hook should advance to the next message. If FALSE, the current message should be returned.

The *pfNoRecord* parameter is the record message flag, which is either TRUE or FALSE. If TRUE, then the message will not be recorded in the JournalRecordHook hook. If FALSE, the message will be recorded in the JournalRecordHook hook.

This function returns either TRUE or FALSE. If TRUE, the pQmsg structure contains the current message to be passed in for handling. If FALSE, The pQmsg structure was not filled in. There are no further messages to pass in.

HK_PLIST_ENTRY - Program List Call Hook

The *program list call hook* is called every time a program-list call or initialization file call is invoked by an application. It is called before the call is run.

This hook, together with its counterpart, ProgramListExitHook, lets applications or system components do the following:

1. Implement the initialization file and program list partially, while retaining the existing implementation. For example, read-only requests could be satisfied from memory, rather than from disk.
2. Redirect initialization-file operations on a particular group to an alternative (opened) profile. For example, in a multiple-user environment, a LAN program might choose to redirect profile groups that are hardware-dependent, rather than user-dependent, to the system-initialization file.

The following code shows the syntax for a program list call hook function:

```
BOOL WINAPI ProgramListEntryHook(HAB hab,
                                 PPRFHOOKPARMS pProfileHookParams,
                                 PBOOL fNoExecute);
```

The *hab* parameter is the anchor block handle.

The *pProfileHookParams* is the profile hook parameters. These identify the call and give its parameters and return code.

The *fNoExecute* parameter is the suppress indicator and is either TRUE or FALSE. If set to TRUE by any hook procedure, no further processing of the call is done. If set to FALSE by all hook procedures, the call is processed normally.

This function returns either TRUE or FALSE. If TRUE, the next hook in the chain is not called. If FALSE, the next hook in the chain is called.

HK_PLIST_EXIT - Program List Exit Hook

The *program list exit hook* called every time a program-list call or initialization file call is invoked by an application. It is called before the call is run. This hook, together with its counterpart, ProgramListEntryHook, lets applications or system components:

1. Implement the initialization file and program list partially, whilst retaining the existing implementation. For example, read-only requests could be satisfied from memory, rather than from disk.
2. Redirect initialization-file operations on a particular group to an alternative (opened) profile. For example, in a multiple-user environment, a LAN program might choose to redirect profile groups that are hardware-dependent, rather than user-dependent, to the system-initialization file.

The following code shows the syntax for a program list exit hook function:

```
BOOL EXPENTRY ProgramListExitHook(HAB hab,  
                                  PPRFHOOKPARMS pProfileHookParams);
```

The *hab* parameter is the anchor block handle.

The *pProfileHookParams* parameter is the profile hook parameters. These identify the call and give its parameters and return code.

This function returns either TRUE or FALSE. If it returns TRUE, the next hook in the chain is not called. If it returns FALSE, the next hook in the chain is called.

HK_REGISTERUSERMSG - Register User Message Hook

The *register user message hook* is called whenever a user message or data type is registered. The following code shows the syntax for a register user message hook function:

```
BOOL EXPENTRY RegisterUserHook(HAB hab,  
                               ULONG cUshort,  
                               PULONG/PUSHORT arRMP,  
                               PBOOL fRegistered);
```

The *hab* parameter is the application anchor block.

The *cUshort* parameter is the number of data type codes. For data types, this parameter is the number of data type codes in the *arRMP* parameter. This value must not be less than one.

For messages, this parameter is set to 0.

The *arRMP* parameter is an array of data type codes. For data types, this parameter is an array of data type codes. For messages, this parameter is set to NULL.

Valid data types are the system-defined data types and their pointer equivalents, application-defined data types and their pointer equivalents, and control data types. Note that not all of the data types that occur in the CPI can be specified in this function.

A control data type is followed by one or more entries in the *arRMP* array that are interpreted in a special way. Control data types allow arrays, offsets, and lengths to be defined. See RegisterUserHook for a full description of the values for the *arRMP* parameter.

The *fRegistered* parameter is the flag indicating that a message or data type was registered. If TRUE, the message or data type was registered. If FALSE, the message or data type was not registered.

The function returns either TRUE (success) or FALSE (an error occurred).

HK_SENDMSG - Send Message Hook

The *send message hook* enables an application to monitor messages that the system does not post to a queue. The system calls a send message hook function while processing WinSendMessage, before delivering the message to the recipient window. By installing an input-hook function and a send message hook function, an application can monitor all window messages effectively. The following code shows the syntax for a send message hook function:

```
VOID WINAPI SendMsgHook(HAB hab,
                        PSMHSTRUCT psmh,
                        BOOL fInterTask);
```

The *psmh* parameter is a pointer to an SMHSTRUCT data structure that contains information about the message.

The *fInterTask* parameter is TRUE if the message is sent between two threads, or FALSE if the message is sent within a thread.

A send message hook function does not return a value, and the next function in the chain is always called. The function can modify values in the SMHSTRUCT data structure before returning.

HK_WINDOWDC - Device Context Hook

The *device context hook* is called when a device context is allocated or freed. The following code shows the syntax for a device context hook function:

```
BOOL WINAPI WindowDCHook(HAB hab,
                          HDC hdc,
                          HWND hwnd,
                          BOOL fAssociate);
```

The *hab* parameter is the application anchor block.

The *hdc* parameter is the current device-context handle.

The *hwnd* parameter is the current window handle.

The *fAssociate* parameter is the association flag and is either TRUE or FALSE. If TRUE, the device context has been allocated. If FALSE, the device context has been freed.

This function returns either TRUE (success) or FALSE (error).

Using Hooks

This section explains how to perform the following tasks:

- Install hook functions
- Release hook functions and free memory
- Record and play back input events

Note: Most of the sample code in this section is part of a complete program which is shown in "Sample Code for Hooks".

Installing Hook Functions

You can install hook functions by calling WinSetHook, specifying the type of hook that calls the function-whether the function is to be associated with the system-message queue or with the queue of a particular thread-and a pointer to a function entry point. The following sample code shows how to install a hook function into the message queue of a thread:

```

BOOL EXPENTRY MyInputHook(HAB, PQMSG, USHORT);
HAB hab;
HMQ hmq;

WinSetHook(hab, /* Anchor block handle */
            hmq, /* Thread message queue */
            HK_INPUT, /* Called by the input hook */
            (PFN) MyInputHook, /* Address of input-hook function */
            (HMODULE) NULL); /* Function is in appl. module */

```

Place hook functions associated with the system-message queue in a dynamic link library (DLL) separate from the application that installs the hook function. The installing application needs the handle of the DLL module before it can install the hook function. DosLoadModule, given the name of the DLL, returns the handle of the DLL module. Once you have the handle, you can call DosQueryProcAddr to obtain the address of the hook function. Finally, use WinSetHook to install the hook-function address in the appropriate hook list. WinSetHook passes the module handle, a pointer to the hook-function entry point, and NULL for the message-queue argument, indicating that the hook function should be associated with the system queue. The following sample code shows functions, called from the application's main routine, that initialize a DLL and install the hook function:

```

HAB habDLL;
HMODULE hMod;
PFN pfnInput;

/*****
/* InitDLL: This function sets up the DLL and sets all variables.
*****/
void EXPENTRY InitDLL(HAB hab)
{
    habDLL = hab;

    /*****
    /* Load the dll - actually, just get our module handle.
    *****/
    DosLoadModule(NULL, 0, "HOOKDLL", &hMod);

    /*****
    /* Find the address of the input hook procedure.
    *****/
    DosQueryProcAddr(hMod, 0, "InputProc", &pfnInput);
}

/*****
/* StartInputHook: This function starts the hook filtering.
*****/
void EXPENTRY StartInputHook(void)
{
    /*****
    /* Set a hook to our input filter routine.
    *****/
    WinSetHook(habDLL, NULLHANDLE, HK_INPUT, pfnInput, hMod);
}

```

Releasing Hook Functions

You can release a hook function and remove its address from the hook list by calling WinReleaseHook with the same arguments that you used when installing the hook function, as shown in the following sample code:

```

BOOL EXPENTRY MyInputHook(HAB, PQMSG, USHORT);
HAB hab;
HMQ hmq;

WinReleaseHook(hab, /* Anchor block handle */

```



```

hmq,                /* Thread message queue          */
HK_INPUT,           /* Called by the input hook       */
(PFN) MyInputHook,  /* Address of input-hook function */
(HMODULE) NULL);    /* Function is in appl. module    */

```

Release all hook functions before the application ends, even though the system automatically releases them if the application does not. You also need to free the memory associated with the hook.

Freeing Memory

How memory for the hook is freed depends on the type of hook chain an event is linked to:

- Queue (current) hook chain
- System hook chain

A queue hook chain is a private hook chain. It applies only to the current calling thread that created the queue with which the hook chain is associated. It may or may not reside in a DLL. If it is not associated with a DLL, its memory can be freed by WinReleaseHook, as shown in the previous sample code.

A system hook chain must reside in a DLL; therefore, it affects the entire system. WinSetHook allocates memory and associates it with a DLL. This memory is not freed until the DLL module is freed. WinReleaseHook cannot free the DLL's memory, because another process cannot free the DLL and its associated memory. However, this memory can be freed by launching a thread that does the following:

- Loads the DLL and sets the hook
- When the playback sequence is complete, releases the hook and frees the DLL, thus relinquishing its memory

As long as any DLL associated with the hook is alive, WinReleaseHook cannot free the memory.

The implication here is straightforward:

- If a queue hook is being installed and it is not associated with a DLL, WinReleaseHook can free its memory.
- If a system hook is being installed, its memory cannot be freed until the DLL is freed. WinReleaseHook has to do a DosFreeModule, but it cannot do this for another process. The application must use DosFreeModule to relinquish hook-allocated memory associated with a DLL.

The following sample code shows a function, called from an application's main routine, that releases the hook and frees the memory of the hook installed in a sample code shown earlier.

```

/*****
/* StopInputHook: This function stops the hook filtering.
*****/
void EXPENTRY StopInputHook(void)
{
/*****
/* Drop a hook to our input filter routine.
*****/
WinReleaseHook(habDLL, NULLHANDLE, HK_INPUT, pfnInput, hMod);

/*****
/* Decrement the DLL usage count.
*****/
DosFreeModule(hMod);
}

```

Recording and Playing Back Input Events

To record and play back input events, use the journal record hook to create a local queue to store the recorded events, then use the journal playback hook to create a second thread to read from the queue. Do not attempt to spend any significant cycles within JournalRecordHook. Because the recorded events include semaphores, Win calls, and I/O functions, it can cause system deadlocks. The following pseudocode describes how to play back recorded functions:

```
Store the passed time as the current time

If the system requests a new message to be prepared (skip is TRUE)
    If all messages have been played back, release the Playback Hook
        (After release, your playback hook function will still
         be called a few times more. So leave a null mouse move
         message as the next message to be copied.)

    Otherwise:
        Save the last message time
        Copy the new message to the passed qmsg buffer
        Calculate the time until the next message
        (You should know, from the recorded times, the
         delta time which actually occurred between each
         message. During playback you will need to calculate
         the amount of time remaining between the time passed
         to you in the qmsg buffer, i.e., "current time", and
         the time at which the next message is due to be
         kicked off.)

Otherwise (skip is FALSE, so the system wants a peek
         at the current message):
    Copy the existing (current) message to the passed qmsg buffer

Recalculate and return the REMAINING delay for the current message
```

An alternative method for installing a system-queue hook function is to provide an installation function in the DLL along with the hook function. With this method, the installing application does not need the handle of the DLL module. By linking with the DLL, the application gains access to the installation function, which can supply the DLL module handle and other details in the call to WinSetHook. The DLL can also contain a function that releases the system-queue hook function. The application can call this hook-releasing function when it ends.

Sample Code for Hooks

This section illustrates a complete hook sample program. Several parts of this program are explained in "Using Hooks".

Hooks Application Sample Code

The hook application includes the following files:

- Hookdemo.C
- Hookdll.C
- Hookdemo.RC
- Hookdemo.H
- Hookdemo.DEF
- Hookdemo.LNK
- Hookdll.DEF
- Hookdll.LNK
- Hookdemo.MAK

The following sample illustrates the hook application code:

```
=====
HOOKDEMO.C
=====
```

[illegible]

```

/*****
/* We must intercept the frame window's messages. */
/* We save the return value (the current WndProc), */
/* so we can pass it all the other messages the frame gets. */
*****/
SysWndProc = WinSubclassWindow(hFrameWnd, (PFNWP)LocalWndProc);

WinShowWindow(hFrameWnd, TRUE);

/*****
/* Standard PM message loop - get it, dispatch it. */
*****/
while (WinGetMsg(hab, &qmsg, NULLHANDLE, 0, 0))
{
    WinDispatchMsg(hab, &qmsg);
}

/*****
/* Clean up on the way out. */
*****/
WinDestroyWindow(hFrameWnd);
WinDestroyMsgQueue(hmq);
WinTerminate(hab);

return TRUE;
}

/*****
/* LocalWndProc() - window procedure for the frame window. */
/* Called by PM whenever a message is sent to the frame. */
*****/
MRESULT EXPENTRY LocalWndProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
    char                szBuffer[80];
    POINTL              pt;
    int                 x;

    switch(msg)
    {

/*****
/* Send the message to the usual WC_FRAME WndProc. */
*****/
        case WM_COMMAND:
            switch (SHORT1FROMMP(mp1))
            {

/*****
/* Start the hook routine - it stops all WM_COMMAND messages. */
/* (which means all these other messages will be ignored). */
*****/
                case IDM_START:
                    StartInputHook();
                    break;

                case IDM_STOP:
                    StopInputHook();
                    break;

                case IDM_EXIT:
                    WinPostMsg(hwnd, WM_CLOSE, 0, 0);
                    break;

                default:
                    return (*SysWndProc)(hwnd, msg, mp1, mp2);
            }
        }
        break;

/*****
/* Send the message to the usual WC_FRAME WndProc. */
*****/
        default:
            return (*SysWndProc)(hwnd, msg, mp1, mp2);
            break;
    }

return FALSE;
}

```

```

=====
HOOKDLL.C
=====
#define INCL_WIN
#define INCL_DOS
#include <os2.h>

/*****
/* Global variables.
*****/
HAB      habDLL;
HMODULE  hMod;
PFN      pfnInput;

/*****
/* InitDLL: This function sets up the DLL and sets all variables
*****/
void EXPENTRY InitDLL(HAB hab)
{
    habDLL = hab;

/*****
/* Load the DLL - actually, just get our module handle.
*****/
    DosLoadModule(NULL, 0, "HOOKDLL", &hMod);

/*****
/* Find the address of the input hook procedure.
*****/
    DosQueryProcAddr(hMod, 0, "InputProc", &pfnInput);
}

/*****
/* StartInputHook: This function starts the hook filtering.
*****/
void EXPENTRY StartInputHook(void)
{
/*****
/* Set a hook to our input filter routine.
*****/
    WinSetHook(habDLL, NULLHANDLE, HK_INPUT, pfnInput, hMod);
}

/*****
/* StopInputHook: This function stops the hook filtering.
*****/
void EXPENTRY StopInputHook(void)
{
/*****
/* Drop a hook to our input filter routine.
*****/
    WinReleaseHook(habDLL, NULLHANDLE, HK_INPUT, pfnInput, hMod);

/*****
/* Decrement the DLL usage count.
*****/
    DosFreeModule(hMod);
}

/*****
/* InputProc: This is the input filter routine.
/* While the hook is active, all messages come here
/* before being dispatched.
*****/
BOOL EXPENTRY InputProc(HAB hab, PQMSG pqMsg, ULONG fs)
{
/*****
/* Check for WM_COMMAND messages.
*****/
    if (pqMsg->msg == WM_COMMAND)
    {
/*****
/* Ignore all WM_COMMAND messages (stops menu processing).
*****/
        return TRUE;
    }
}

```

```

    }

/*****
/*  Pass the message on to the next hook in line.  */
*****/
    return FALSE;
}

=====
HOOKDEMO.RC
=====
#include <os2.h>
#include "hookdemo.h"

MENU      HOOKDEMO
BEGIN
    SUBMENU      "Command",   IDM_CMD
    BEGIN
        MENUITEM  "Start",     IDM_START
        MENUITEM  "Stop",      IDM_STOP
        MENUITEM  "Exit",      IDM_EXIT
    END
END

=====
HOOKDEMO.H
=====
#define HOOKDEMO      256
#define IDM_CMD       400
#define IDM_START     401
#define IDM_STOP      402
#define IDM_EXIT      403

=====
HOOKDEMO.DEF
=====
NAME HOOKDEMO WINDOWAPI

DESCRIPTION 'PM Hooks Sample'

CODE  MOVEABLE
DATA  MOVEABLE MULTIPLE

STACKSIZE  24576
HEAPSIZE   10240

PROTMODE

=====
HOOKDEMO.LNK
=====
hookdemo.obj /NOI
hookdemo.exe
hookdemo.map
hookdll.lib
hookdemo.def

=====
HOOKDLL.DEF
=====
LIBRARY HOOKDLL

DESCRIPTION 'PM Hooks Sample'

CODE  LOADONCALL
DATA  LOADONCALL

PROTMODE

EXPORTS
    InitDLL
    StartInputHook
    StopInputHook
    InputProc

=====
HOOKDLL.LNK
=====
hookdll.obj /NOI

```

```

hookdll.dll
hookdll.map
hookdll.def

=====
HOOKDEMO.MAK
=====
CC      = icc /c /Ge /Gd- /Se /Re /ss /Gm+
LINK    = link386
HEADERS = hookdemo.h

#-----
#  A list of all of the object files.
#-----
ALL_OBJ1 = hookdemo.obj

ALL_OBJ2 = hookdll.obj

all: hookdemo.exe hookdll.dll

hookdemo.res: hookdemo.rc hookdemo.h

hookdemo.obj: hookdemo.c $(HEADERS)
               icc /C /Ss /W3 hookdemo.c

hookdll.obj:   hookdll.c
               icc /C+ /Ge- /Gm+ hookdll.c

hookdll.dll:   $(ALL_OBJ2) hookdll.def hookdll.lnk
               $(LINK) @hookdll.lnk
               implib hookdll.lib hookdll.def

hookdemo.exe:  $(ALL_OBJ1) hookdemo.def hookdemo.lnk hookdemo.res hook dll.lib
               $(LINK) @hookdemo.lnk
               rc -p -x hookdemo.res hookdemo.exe

```

Initialization Files

Initialization files enable an application to store and retrieve information that the application uses when it starts up. This chapter describes how to use the IBM OS/2 Profile Manager to create, manage, and use the system's initialization files. The following topics are related to this chapter:

- File system
- Presentation Manager interface applications

About Initialization Files

An initialization file is a convenient place to store information between sessions. Profile Manager enables applications to create their own initialization files and to access the OS/2 initialization files, *os2.ini* and *os2sys.ini*. Just as the system uses the *os2.ini* and *os2sys.ini* files to store configuration information for system startup, an application can create an initialization file that stores information it uses to initialize windows and data.

The system initialization files contain sections and settings used by the PM applications. Although applications can read settings from the initialization files, only rarely does an application need to change a setting. OS/2 initialization files are binary; the user cannot view or edit them directly.

An initialization file consists of one or more sections; each section contains one or more settings, or keys. Each key consists of two parts: a name and a value. Both section names and key names are null-terminated strings. The value assigned to a key can be a null-terminated string, a null-terminated string representing a signed integer, or individual bytes of data.

Once an initialization file is created, an application can rename, copy, move, or delete that file just as it does any other file. Although an

application also could read directly to or write directly to the initialization file, the application should always use Profile Manager functions to access the contents of the file. Both character-based OS/2 applications and PM applications can use Profile Manager functions. Before calling Profile Manager, a thread must initialize an anchor block by using the WinInitialize function.

Using Initialization Files

This section explains how to use Profile Manager functions to perform the following tasks:

- Create, open, and close initialization files
- Read and write settings
- Identify the initialization files

Creating, Opening, and Closing Initialization Files

You can create an initialization file or open an existing initialization file by using the PrfOpenProfile function. The function requires a handle to an anchor block and a pointer to the name of an initialization file. If the file does not exist in the given path, the function automatically creates an initialization file.

The following code fragment creates an initialization file named *pmtools.ini* in the current directory:

```
HAB hab;
HINI hini;

hab = WinInitialize(0);
if ((hini = PrfOpenProfile(hab, "pmtools.ini")) == NULL){
    .
    . /* File was not created */
    .
}
```

If the PrfOpenProfile function is successful, it returns a handle to the initialization file. Otherwise, it returns NULL, and the file is not created. Once you have an initialization-file handle, you can create new sections and settings in the file.

To close an initialization file, you use the PrfCloseProfile function.

Reading and Writing Settings

An application can store strings, integers, and binary data in an initialization file and retrieve them. To read from or write to an initialization file, your application must provide a section name and a key name that specify which setting to read or change. If the section or key name you specify in a writing operation does not exist in the file, it is added to the file and assigned the given value.

The following code fragment creates a section named "MyApp" and a key named "MainWindowColor" in a previously opened initialization file, and assigns the value of the RGB structure to the new setting:

```
HINI hini;
RGB rgb = { 0xff, 0x00, 0x00 };

PrfWriteProfileData(hini, "MyApp", "MainWindowColor", &rgb, sizeof(RGB));
```


To read a setting, your application can retrieve the size of the setting and then read the setting into an appropriate buffer by using the `PrfQueryProfileSize` and `PrfQueryProfileData` functions, as shown in the following example. This example reads the setting "MainWindowColor" from the "MyApp" section only if the size of the data is equal to the size of the RGB structure.

```
HINI hini;
ULONG cb;
RGB rgb;

PrfQueryProfileSize(hini, "MyApp", "MainWindowColor", &cb);
if (cb == sizeof(RGB))
    PrfQueryProfileData(hini, "MyApp", "MainWindowColor", &rgb, &cb);
```

An application can also read strings by using the `PrfQueryProfileString` function, write strings by using the `PrfWriteProfileString` function, and read integers (stored as strings) by using the `PrfQueryProfileInt` function.

Identifying the OS/2 Initialization Files

Your application can retrieve the names of the system initialization files by using the `PrfQueryProfile` function. Although the OS/2 initialization files are usually named `os2.ini` and `os2sys.ini`, you can use other files when starting the system.

The following example retrieves the names of the initialization files and copies their names to the strings `szUserName` and `szSysName`. Once you know the names of the OS/2 initialization files, you can use them to open the files and read settings.

```
CHAR szUserName[CCHMAXPATH];
CHAR szSysName[CCHMAXPATH];
HINI hini;

PRFPROFILE prfpro = { sizeof(szUserName), szUserName,
                     sizeof(szSysName), szSysName };

PrfQueryProfile(hini, &prfpro);
```

You can change the OS/2 initialization files to files of your choice by using the `PrfReset` function. This function requires the names of two initialization files and uses them as replacements for the `os2.ini` and `os2sys.ini` files. The system is then reset by using the settings in the new files.

Keyboard Accelerators

A *keyboard accelerator* (*shortcut key* to the user) is a keystroke that generates a command message for an application. This chapter describes how to use keyboard accelerators in your PM applications.

About Keyboard Accelerators

Using a keyboard accelerator has the same effect as choosing a menu item. While menus provide an easy way to learn an application's command set, accelerators provide quick access to those commands.

Without accelerators, a user might generate commands by pressing the Alt key to access the menu bar, using the Arrow keys to select an item, then pressing the Enter key to choose the item. In contrast, accelerators allow the user to generate commands *with a single* keystroke.

Like menu items, accelerators can generate WM_COMMAND, WM_HELP, and WM_SYSCOMMAND messages. Although, normally, accelerators are used to generate existing commands as menu items, they also can send commands that have no menu-item equivalent.

Accelerator Tables

An accelerator table contains an array of accelerators. Accelerator tables exist at two levels within the operating system: a single accelerator table for the system queue and individual accelerator tables for application windows. Accelerators in the system queue apply to all applications—for example, the F1 key always generates a WM_HELP message. Having accelerators for individual application windows ensures that an application can define its own accelerators without interfering with other applications. An accelerator for an application window can override the accelerator in the system queue. An application can modify both its own accelerator table and the system's accelerator table.

The application can set and query the accelerator table for a specific window or for the entire system. For example, an application can query the system accelerator table, copy it, modify the copied data structures; and then, use the modified copy to set the system accelerator table. An application also can modify its window's accelerator table at run time to respond more appropriately to the current environment.

Note: An application that modifies any accelerator table other than its own should maintain the original accelerator table; and, before terminating, restore that table.

Accelerator-Table Resources

You can use accelerators in an application by creating an accelerator-table resource in a resource-definition file. Then, when the application creates a standard frame window, the application can associate that window with the resource.

As specified in a resource-definition file, an accelerator table consists of a list of accelerator items, each defining the keystroke that triggers the accelerator, the command the accelerator generates, and the accelerator's style. The style specifies whether the keystroke is a virtual key, a character, or a scan code, and whether the generated message is WM_COMMAND, WM_SYSCOMMAND, or WM_HELP; WM_COMMAND is the default.

Accelerator-Table Handles

Applications that use accelerator tables refer to them with a 32-bit handle. An application using this handle, by default, can make most API function calls for accelerators without having to account for the internal structures that define the accelerator table. When an application needs to dynamically create or change an accelerator table, it must use the ACCEL and ACCELTABLE data structures.

Accelerator-Table Data Structures

An accelerator table consists of individual accelerator items. Each item in the table is represented by an ACCEL structure that defines the accelerator's style, keystroke, and command identifier. Typically, an application defines these aspects of an accelerator in a resource-definition file, but the ACCEL structure also can be built in memory at run time. An accelerator table is represented by an ACCELTABLE structure that specifies the number of accelerator items in the table, the code page used for the keystrokes in the accelerator items, and an array of ACCEL structures (one for each item in the table). Applications that use ACCELTABLE structures directly must allocate sufficient memory to hold all the items in the table.

Accelerator-Item Styles

An accelerator item has a style that determines what combination of keys produces the accelerator and what command message is generated by the accelerator. An application can specify the following accelerator-item styles in the *fs* field of the ACCEL structure:

Style	Description
AF_ALT	Specifies that the user must hold down the Alt key while pressing the accelerator key.
AF_CHAR	Specifies that the keystroke is a character that is translated using the code page for the accelerator table. (This is the default style.)
AF_CONTROL	Specifies that the user must hold down the Ctrl key while pressing the accelerator key.
AF_HELP	Specifies that the accelerator generates a WM_HELP message instead of a WM_COMMAND message.
AF_LONEKEY	Specifies that the user need not press another key while the accelerator key is down. Typically, this style is used with the Alt key to specify that simply pressing and releasing that key triggers the accelerator.
AF_SCANCODE	Specifies that the keystroke is an untranslated scan code from the keyboard.
AF_SHIFT	Specifies that the user must hold down the Shift key when pressing the accelerator key.
AF_SYSCOMMAND	Specifies that the accelerator generates a WM_SYSCOMMAND message instead of a WM_COMMAND message.
AF_VIRTUALKEY	Specifies that the keystroke is a virtual key—for example, the F1 function key.

Using Keyboard Accelerators

This section explains how to perform the following tasks:

- Create an accelerator-table resource
- Include an accelerator table in a frame window
- Modify an accelerator table

Creating an Accelerator-Table Resource

The following code fragment shows a typical accelerator-table resource:

```
ACCELTABLE ID_ACCEL_RESOURCE
BEGIN
    VK_ESC,      IDM_ED_UNDO,  AF_VIRTUALKEY | AF_SHIFT
```

```

VK_DELETE, IDM_ED_CUT,    AF_VIRTUALKEY
VK_F2,      IDM_ED_COPY,  AF_VIRTUALKEY
VK_INSERT,  IDM_ED_PASTE, AF_VIRTUALKEY
END

```

This accelerator table has four accelerator items. The first one is triggered when the user presses Shift+Esc, which sends a WM_COMMAND message (the default).

An accelerator table in a resource-definition file has an identifier (ID_ACCEL_RESOURCE in the previous example). You can associate an accelerator-table resource with a standard frame window by specifying the table's resource identifier as the *idResources* parameter of the WinCreateStdWindow function.

An application can load an accelerator table resource-definition file automatically when creating a standard frame window, or it can load the resource independently and associate it with a window or with the entire system.

Including an Accelerator Table in a Frame Window

You can add an accelerator table to a frame window either by using the WinSetAccelTable function or by defining an accelerator-table resource (as shown in the previous section) and creating a frame window with the FCF_ACCELTABLE frame style. The second method is shown in the following code fragment:

```

HWND  hwndFrame,hwndClient;
CHAR  szClassName[ ]="MyClass";
CHAR  szTitle[ ]="MyWindow";

ULONG flControlStyle=
    FCF_MINMAX           |           /* Min and max buttons */
    FCF_SHELLPOSITION    |           /* System size and position */
    FCF_SIZEBORDER        |           /* Size border */
    FCF_TITLEBAR          |           /* Title bar */
    FCF_TASKLIST          |           /* Task list */
    FCF_ACCELTABLE        |           /* Accelerator table */
    FCF_SYSMENU           |           /* System menu */
    FCF_MENU;             |           /* Menu */

hwndFrame=WinCreateStdWindow(HWND_DESKTOP,
    WS_VISIBLE,
    &flControlStyle,
    szClassName,
    szTitle,
    0,
    (HMODULE)NULL,
    ID_MENU_RESOURCE,
    &hwndClient);

```

Notice that if you set the *flControlStyle* parameter to the FCF_STANDARD flag, you must define an accelerator-table resource, because FCF_STANDARD includes the FCF_ACCELTABLE flag.

If the window being created also has a menu, the menu resource and accelerator resource must have the same resource identifier; this is because the WinCreateStdWindow function has only one input parameter to specify the resource identifiers for menus, accelerator tables, and icons. If an application creates an accelerator table resource-definition file; then, opens a standard frame window (as shown in the preceding example), the accelerator table is installed automatically in the window's message queue, and keyboard events are translated during the normal processing of events. The application simply responds to WM_COMMAND, WM_SYSCOMMAND, and WM_HELP messages; it does not matter whether these messages come from a menu or an accelerator.

An application also can add an accelerator table to a window by calling the WinSetAccelTable function with an accelerator-table handle and a frame-window handle. The application can call either the WinLoadAccelTable function to retrieve an accelerator table from a resource file or the WinCreateAccelTable function to create an accelerator table from an accelerator-table data structure in memory.

Modifying an Accelerator Table

You can modify an accelerator table, for either your application windows or the system, by doing the following:

1. Retrieve the handle of the accelerator table.
2. Use that handle to copy the accelerator-table data to an application-supplied buffer.
3. Change the data in the buffer.
4. Use the changed data to create a new accelerator table.

Then you can use the new accelerator-table handle to set the accelerator table, as outlined in the following list:

1. Call WinQueryAccelTable to retrieve an accelerator-table handle.
2. Call WinCopyAccelTable with a NULL buffer handle to determine how many bytes are in the table.
3. Allocate sufficient memory for the accelerator-table data.
4. Call WinCopyAccelTable, with a pointer to the allocated memory.
5. Modify the data in the buffer (assuming it has the form of an ACCELTABLE structure).
6. Call WinCreateAccelTable, passing a pointer to the buffer with the modified accelerator-table data.
7. Call WinSetAccelTable with the handle returned by WinCreateAccelTable.

List-Box Controls

A *list box* is a control window that displays several text items at a time, one or more of which can be selected by the user. This chapter explains how to create and use list-box controls in PM applications.

About List Boxes

An application uses a list box when it requires a list of selectable fields that is too large for the display area or a list of choices that can change dynamically. Each list item contains a text string and a handle. Usually, the text string is displayed in the list-box window; but the handle is available to the application to reference other data associated with each of the items in the list.

A list box always is owned by another window that receives messages from the list box when events occur, such as when a user selects an item from the list box. Typically, the owner is a dialog window (as shown in the following figure,) or the client window of an application frame window. The client- or dialog-window procedure defined by the application responds to messages sent from the list box.

A list box always contains a scroll bar for use when the list box contains more items than can be displayed in the list-box window. The list box responds to mouse clicks in the scroll bar by scrolling the list; otherwise, the scroll bar is disabled.

The maximum number of items permitted in a list box is 32767.

Using List Boxes

An application uses a list-box control to display a list in a window. List boxes can be displayed in standard application windows, although they are more commonly used in dialog windows. In either case, notification messages are sent from the list box to its owner window, enabling the application to respond to user actions in the list.

Once a list box is created, the application controls the insertion and deletion of list items. Items can be inserted at the end of the list, automatically sorted into the list, or inserted at a specified index position. Applications can turn list drawing on and off to speed up the process of inserting numerous items into a list.

The owner-window procedure of the list box receives messages when a user manipulates the list-box data. Most default list actions (for example, highlighting selections and scrolling) are handled automatically by the list box itself. The application controls the responses when the user chooses an item in the list, either by double-clicking the item or by pressing Enter after an item is highlighted. The list box also notifies the application when the user changes the selection or scrolls the list.

Normally, list items are text strings drawn by a list box. An application also can draw and highlight the items in a list. This enables the application to create customized lists that contain graphics. When an application creates a list box with the LS_OWNERDRAW style, the owner of the list box receives a WM_DRAWITEM message for each item that should be drawn or highlighted. This is similar to the owner-drawn style for menus, except that the owner-drawn style applies to the entire list rather than to individual items.

Creating a List-Box Window

List boxes are WC_LISTBOX class windows and are predefined by the system. Applications can create list boxes by calling WinCreateWindow, using WC_LISTBOX as the window-class parameter.

A list box passes notification messages to its owner window, so an application uses its client window, rather than the frame window, as the owner of the list. The client-window procedure receives the messages sent from the list box. For example, to create a list box that completely fills the client area of a frame window, an application would make the client window the owner and parent of the list-box window, and make the list-box window the same size as the client window. This is shown in the following code fragment.

```
#define ID_LISTWINDOW    250

HWND hwndClient, hwndList;
RECT rcl;

/* How big is the
   client window? */
WinQueryWindowRect(hwndClient, &rcl);

/* Make a list-box
   window. */
hwndList = WinCreateWindow(hwndClient,
    WC_LISTBOX,
    "",
    WS_VISIBLE | LS_NOADJUSTPOS,
    0, 0,
    rcl.xRight, rcl.yTop,
    hwndClient,
    HWND_TOP,
    ID_LISTWINDOW,
    NULL,
    NULL);
/* Parent
   /* Class
   /* Name
   /* Style
   /* x, y
   /* cx, cy
   /* Owner
   /* Behind
   /* ID
   /* Control data
   /* parameters
```

Because the list box draws its own border, and a frame-window border already surrounds the client area of a frame window due to the adjacent frame controls, the effect is a double-thick border around the list box. You can change this effect by calling WinInflateRect to overlap the list-box border with the surrounding frame-window border, resulting in only one list-box border.

Notice that the code specifies the list-box window style LS_NOADJUSTPOS. This ensures that the list box is created exactly the specified size. If the LS_NOADJUSTPOS style is not specified, the list-box height is rounded down, if necessary, to make it a multiple of the item height. Enabling a list box to adjust its height automatically is useful for preventing partial items being displayed at the bottom of a list box.

Using a List Box in a Dialog Window

List boxes most commonly are used in dialog windows. A list box in a dialog box is a control window, like a push button or an entry field. Typically, the application defines a list box as one item in a dialog template in the resource-definition file, as shown in the following resource compiler source-code fragment.

```
DLGTEMPLATE IDD_OPEN
```

```

BEGIN
    DIALOG "Open...", IDD_OPEN, 35, 35, 150, 135,
        FS_DLGGBORDER, FCF_TITLEBAR
    BEGIN
        LISTBOX        IDD_FILELIST, 15, 15, 90, 90
        PUSHBUTTON     "Drive", IDD_DRIVEBUTTON, 115, 70, 30, 14
        DEFPUSHBUTTON  "Open", IDD_OPENBUTTON, 115, 40, 30, 14
        PUSHBUTTON     "Cancel", IDD_CANCELBUTTON, 115, 15, 30, 14
    END
END

```

Once the dialog resource is defined, the application loads and displays the dialog box as it would normally. The application inserts items into the list when processing the WM_INITDLG message.

A dialog window with a list box usually has an **OK** button. The user can select items in the list, and then indicate a final selection by double-clicking, pressing Enter, or clicking the **OK** button. When the dialog-window procedure receives a message indicating that the user has clicked the **OK** button, it queries the list box to determine the current selection (or selections, if the list allows multiple selections), and then responds as though it had received a WM_CONTROL message with the LN_ENTER notification code.

Adding or Deleting an Item in a List Box

Applications can add items to a list box by sending an LM_INSERTITEM or LM_INSERTMULTITEMS message to the list-box window; items are deleted using the LM_DELETEITEM message. Items in a list are specified with a 0-based index (beginning at the top of the list). A new list is created empty; the application initializes the list by inserting items. LM_INSERTMULTITEMS allows up to 32767 items to be inserted as a group, while LM_INSERTITEM adds items one-by-one to a list.

The application specifies the text and position for each new item. It can specify an *absolute-position* index or one of the following predefined index values:

Value	Meaning
<code>LIT_END</code>	Insert item at end of list.
<code>LIT_SORTASCENDING</code>	Insert item alphabetically ascending into list.
<code>LIT_SORTDESCENDING</code>	Insert item alphabetically descending into list.

If a large number of items are to be inserted into a list box at one time, use of LM_INSERTMULTITEMS is more efficient than use of LM_INSERTITEM. The same positioning flags are used. When LIT_SORTASCENDING or LIT_SORTDESCENDING is specified with LM_INSERTMULTITEMS, new items are inserted before the updated list is sorted. If items are being added using several LM_INSERTMULTITEMS messages, LIT_END should be specified for all messages except the last; this will avoid unnecessary multiple sorts of the list.

If no text array is specified, empty items are inserted into the list. This is very useful for list boxes created with LS_OWNERDRAW style, which do not use text strings.

The application must send an LM_DELETEITEM message and supply the absolute-position index of the item when deleting items from a list. The LM_DELETEALL message deletes all items in a list.

One way an application can speed up the insertion of list items is to suspend drawing until it has finished inserting items. This is a particularly valuable approach when using a sorted insertion process (when inserting one item can cause rearrangement of the entire list). You can turn off list drawing by calling WinEnableWindowUpdate, specifying FALSE for the *enable* parameter, and then calling WinShowWindow. This forces a total update when insertion is complete. The following code fragment illustrates this concept:

```

HWND hwndFileList;

/* Disable updates while filling the list. */
WinEnableWindowUpdate(hwndFileList, FALSE);
.
. /* Send LM_INSERTITEM messages to insert all new items. */
.

```

```
/* Now cause the window to update and show the new information. */
WinShowWindow(hwndFileList, TRUE);
```

Notice that this optimization is unnecessary if an application is adding list items while processing a WM_INITDLG message, because the list box is not visible, and the list-box routines are internally optimized.

Responding to a User Selection in a List Box

When a user chooses an item in a list, the primary notification an application receives is a WM_CONTROL message, with the LN_ENTER control code sent to the owner window of the list. Within the window procedure for the owner window, the application responds to the LN_ENTER control code by querying the list box for the current selection (or selections, in the case of an LS_MULTIPLESEL or LS_EXTENDEDSEL list box).

The LN_ENTER control code notifies the application that the user has selected a list item. A WM_CONTROL message with an LN_SELECT control code is sent to the list-box owner whenever a selection in a list changes, such as when a user moves the mouse pointer up and down a list while pressing the mouse button. In this case, items are selected but not yet *chosen*. An application can ignore LN_SELECT control codes when the selection changes, responding only when the item is actually chosen. Or an application can use LN_SELECT to display context-dependent information that changes rapidly with each selection made by the user.

Handling Multiple Selections

When a list box has the style LS_MULTIPLESEL or LS_EXTENDEDSEL, the user can select more than one item at a time. An application must use different strategies when working with these types of lists. For example, when responding to an LN_ENTER control code, it is not sufficient to send a single LM_QUERYSELECTION message, because that message will find only the first selection. To find all current selections, an application must continue sending LM_QUERYSELECTION messages, using the return index of the previous message as the starting index of the next message, until no items are returned.

Creating an Owner-Drawn List Item

To draw its own list items, an application must create a list that has the style LS_OWNERDRAW: the owner window of the list box must respond to the WM_MEASUREITEM and WM_DRAWITEM messages.

When the owner window receives a WM_MEASUREITEM message, it must return the height of the list item. All items in a list must have the same height (greater than or equal to 1). The WM_MEASUREITEM message is sent when the list box is created, and every time an item is added. You can change the item height by sending an LM_SETITEMHEIGHT message to the list-box window. The maximum width of a list box created with the LM_HORZSCROLL style can be set using an LM_SETITEMWIDTH message.

The owner window receives a WM_DRAWITEM message whenever an item in an owner-drawn list should be drawn or highlighted. Although it is quite common for an owner-drawn list to draw items, it is less common to override the system-default method of highlighting. (This method inverts the rectangle that contains the item.) Do not create your own highlighting unless, for some reason, the system-default method is unacceptable to you.

The WM_DRAWITEM message contains a pointer to an OWNERITEM data structure. The OWNERITEM structure contains the window identifier for the list box, a presentation-space handle, a bounding rectangle for the item, the position index for the item, and the application-defined item handle. This structure also contains two fields that determine whether a message draws, highlights, or removes the highlighting from an item. The OWNERITEM structure has the following form:

```
typedef struct _OWNERITEM { /* oi */
    HWND    hwnd;
    HPS     hps;
    ULONG    fsState;
    ULONG    fsAttribute;
    ULONG    fsStateOld;
    ULONG    fsAttributeOld;
    RECT     rclItem;
    LONG     idItem;
```



```

        ULONG    hItem;
    } OWNERITEM;

```

When the item must be drawn, the owner window receives a WM_DRAWITEM message with the *fsState* field set differently from the *fsStateOld* field. If the owner window draws the item in response to this message, it returns TRUE, telling the system not to draw the item. If the owner window returns FALSE, the system draws the item, using the default list-item drawing method.

You can get the text of a list item by sending an LM_QUERYITEMTEXT message to the list-box window. You should draw the item using the *hps* and *rcItem* arguments provided in the OWNERITEM structure.

If the item being drawn is currently selected, the *fsState* and *fsStateOld* fields are both TRUE; they both will be FALSE if the item is not currently selected. The window receiving a WM_DRAWITEM message can use this information to highlight the selected item at the same time it draws the item. If the owner window highlights the item, it must leave the *fsState* and *fsStateOld* fields equal to each other. If the system provides default highlighting for the item (by inverting the item rectangle), the owner window must set the *fsState* field to 1 and the *fsStateOld* field to 0 before returning from the WM_DRAWITEM message.

The owner window also receives a WM_DRAWITEM message when the highlight state of a list item changes. For example, when a user clicks an item, the highlighting must be removed from the currently selected item, and the new selection must be highlighted. If these items are owner-drawn, the owner window receives one WM_DRAWITEM message for each unhighlighted item and one message for the newly highlighted item. To highlight an item, the *fsState* field must equal TRUE, and the *fsStateOld* field must equal FALSE. In this case, the application should highlight the item and return the *fsState* and *fsStateOld* fields equal to FALSE, which tells the system not to highlight the item. The application also can return the *fsState* and *fsStateOld* fields with two different (unequal) values and the list box will highlight the item (the default action).

To remove highlighting from an item, the *fsState* field must equal FALSE and the *fsStateOld* field must equal TRUE. In this case, the application removes the highlighting and returns both the *fsState* and the *fsStateOld* fields equal to FALSE. This tells the system not to attempt to remove the highlighting. The application also can return the *fsState* and *fsStateOld* fields with two different (unequal) values, and the list box will remove the highlighting (the default response). The following code fragment shows these selection processes:

```

OWNERITEM *poi;

case WM_DRAWITEM:

    /* Convert mp2 into an OWNERITEM structure pointer. */
    poi = (OWNERITEM) PVOIDFROMMP(mp2);

    /* Test to see if this is drawing or highlighting/unhighlighting. */
    if (poi->fsState != poi->fsStateOld) {

        /* This is either highlighting or unhighlighting. */
        if (poi->fsState) {
            . /* Highlight the item. */
            .
        }
        else {
            . /* Remove the highlighting. */
            .
        }

        /* Set fsState = fsStateOld to tell system you did it. */
        poi->fsState = poi->fsStateOld = 0;

        return TRUE; /* Tells list box you did the highlighting. */
    }
    else {
        . /* Draw the item. */
        .
        if (poi->fsState) { /* Checks to see if item is selected */
            . /* Highlight the item. */
            .
            /* Set fsState = fsStateOld to tell system you did it. */
        }
        return TRUE; /* Tells list box you did the drawing. */
    }
}

```

Default List-Box Behavior

The following table lists all the messages handled by the predefined list-box window-class procedure.

Message	Description
<code>LM_DELETEALL</code>	Deletes all items in the list.
<code>LM_DELETEITEM</code>	Removes the specified item from the list, redrawing the list as necessary. Returns the number of items remaining in the list.
<code>LM_INSERTITEM</code>	Inserts a new item into the list according to the position information passed with the message.
<code>LM_INSERTMULTITEMS</code>	Inserts one or more items into a list box at one time.
<code>LM_QUERYITEMCOUNT</code>	Returns the number of items in the list.
<code>LM_QUERYITEMHANDLE</code>	Returns the specified item handle.
<code>LM_QUERYITEMTEXT</code>	Copies the text of the specified item to a buffer supplied by the message sender.
<code>LM_QUERYITEMTEXTLENGTH</code>	Returns the text length of the specified item.
<code>LM_QUERYSELECTION</code>	For a single-selection list box, returns the zero-based index of the currently selected item. For a multiple-selection list box, returns the next selected item or <code>LIT_NONE</code> if no more items are selected.
<code>LM_QUERYTOPINDEX</code>	Returns the zero-based index to the item currently visible at the top of the list.
<code>LM_SEARCHSTRING</code>	Searches the list for a match to the specified string.
<code>LM_SELECTITEM</code>	Selects the specified item. If the list is a single-selection list, deselects the previous selection. Sends a <code>WM_CONTROL</code> message (with the <code>LN_SELECT</code> code) to the owner window.
<code>LM_SETITEMHANDLE</code>	Sets the specified item handle.
<code>LM_SETITEMHEIGHT</code>	Sets the item height for the list. All items in the list have the same height.
<code>LM_SETITEMTEXT</code>	Sets the text for the specified item.
<code>LM_SETITEMWIDTH</code>	Sets the maximum width of a list box created with the <code>LS_HORZSCROLL</code> style.
<code>LM_SETTOPINDEX</code>	Shows the specified item as the top item in the list window, scrolling the list as necessary.
<code>WM_ADJUSTWINDOWPOS</code>	If the list box has the style <code>LS_NOADJUSTPOS</code> , makes no changes to

	the SWP structure and returns FALSE. Otherwise, adjusts the height of the list box so that a partial item is not shown at the bottom of the list. Returns TRUE if the SWP structure is changed.
WM_BUTTON2DOWN	Returns TRUE; the message is ignored.
WM_BUTTON3DOWN	Returns TRUE; the message is ignored.
WM_CHAR	Processes virtual keys for line and page scrolling. Sends an LN_ENTER notification code for the Enter key. Returns TRUE if the key is processed; otherwise, passes the message to the WinDefWindowProc function.
WM_CREATE	Creates an empty list box with a scroll bar.
WM_DESTROY	Destroys the list and deallocates any memory allocated during its existence.
WM_ENABLE	Enables the scroll bar if there are more items than can be displayed in a list-box window.
WM_MOUSEMOVE	Sets the mouse pointer to the arrow shape and returns TRUE to show that the message was processed.
WM_PAINT	Draws the list box and its items.
WM_HSCROLL	Handles scrolling indicated by the list-box horizontal scroll bar.
WM_VSCROLL	Handles scrolling indicated by the list-box vertical scroll bar.
WM_SETFOCUS	If the list box is gaining the focus, creates a cursor and sends an LN_SETFOCUS notification code to the owner window. If the list box is losing the focus, this message destroys the cursor and sends an LN_KILLFOCUS notification code to the owner window.
WM_TIMER	Uses timers to control automatic scrolling that occurs when a user drags the mouse pointer outside the window.

Menus

A *menu* is a window that contains a list of items-text strings, bit maps, or images drawn by the application-that enables the user, by mouse or keyboard, to choose from these predetermined choices. This chapter describes how to use menus in your PM applications.

About Menus

A menu always is owned by another window, usually a frame window. When a user makes a choice from a menu, the menu posts a message containing the unique identifier for the menu item to its owner by way of the owner window's window procedure.

An application typically defines its menus using Resource Compiler, and then associates the menus with a frame window when the frame window is created. Applications also can create menus by filling in menu-template data structures and creating windows with the WC_MENU class. Either way, applications can add, delete, or change menu items dynamically by issuing messages to menu windows.

Menu Bar and Pull-Down Menus

A typical application uses a menu bar and several pull-down submenus. The pull-down submenus ordinarily are hidden, but become visible when the user makes selections in the menu bar. Pull-down submenus always are attached to the menu bar.

The menu bar is a child of the frame window; the menu bar window handle is the key to communicating with the menu bar and its submenus. You can retrieve this handle by calling WinWindowFromID, with the handle of the parent window and the FID_MENU frame-control identifier. Most messages for the menu bar and its submenus can be issued to the menu-bar window. Flags in the messages tell the window whether to search submenus for requested menu items.

Pop-Up Menus

A pop-up menu is like a pull-down submenu, except that it is not attached to the menu bar; it can appear anywhere in its parent window. A pop-up menu usually is associated with a *portion* of a window, such as the client window, or it is associated with a specific object, such as an icon.

A pop-up menu remains hidden until the user selects it (either by moving the cursor to the appropriate location and pressing Enter or clicking on the location with the mouse). Typically, pop-up menus are displayed at the position of the cursor or mouse pointer; they provide a quick mechanism for selecting often-used menu items.

To include a pop-up menu in an application, you first must define a menu resource in a resource-definition file, then load the resource using the WinLoadMenu or WinCreateMenu functions. You must call WinPopupMenu to create the pop-up menu and display it in the parent window. Applications typically call WinPopupMenu in a window procedure in response to a user-generated message, such as WM_BUTTON2DBLCLK or WM_CHAR.

WinPopupMenu requires that you specify the pop-up menu's handle and also the handles of the parent and owner windows of the pop-up menu. WinLoadMenu and WinCreateMenu return the handle of the pop-up menu window, but you must obtain the handles of the parent and owner by using WinQueryWindow.

You determine the position of the pop-up menu in relation to its parent by specifying coordinates and style flags in WinPopupMenu. The *x* and *y* coordinates determine the position of the lower-left corner of the menu relative to the lower-left corner of the parent. The system may adjust this position, however, if you include the PU_HCONSTRAIN or PU_VCONSTRAIN style flags in the call to WinPopupMenu. If necessary, PU_HCONSTRAIN adjusts the horizontal position of the menu so that its left and right edges are within the borders of the desktop window. PU_VCONSTRAIN makes the same adjustments vertically. Without these flags, a desktop-level pop-up menu can lie partially off the screen, with some items not visible nor selectable.

The PU_POSITIONONITEM flag also can affect the position of the pop-up menu. This flag positions the pop-up menu so that, when the pop-up menu appears, the specified item lies directly under the mouse pointer. Also, PU_POSITIONONITEM automatically selects the item. PU_POSITIONONITEM is useful for placing the current menu selection under the pointer so that, if the user releases the mouse button without selecting a new item, the current selection remains unchanged.

The PU_SELECTITEM flag is similar to PU_POSITIONONITEM except that it just selects the specified item; it does not affect the position of the menu.

You can enable the user to choose an item from a pop-up menu by using the same mouse button that was used to display the menu. To do this, specify the PU_MOUSEBUTTON*n* flag, where *n* corresponds to the mouse button used to display the menu. This flag specifies the mouse buttons for the user to interact with a pop-up menu once it is displayed.

By using the PU_MOUSEBUTTON*n* flag, you can enable the user to display the pop-up menu, select an item, and dismiss the menu, all in one operation. For example, if your window procedure displays the pop-up window when the user double-clicks mouse button 2, specify the PU_MOUSEBUTTON2DOWN flag in the WinPopupMenu function. Then, the user can display the menu with mouse button 2; and, while holding the button down, select an item. When the user releases the button, the item is chosen and the menu dismissed.

System Menu

The system menu in the upper-left corner of a standard frame window is different from the menus defined by the application. The system menu is controlled and defined almost exclusively by the system; your only decision about it is whether to include it when creating a frame window. (It is unusual for a frame window *not* to include a system menu.) The system menu generates WM_SYSCOMMAND messages instead of WM_COMMAND messages. Most applications simply use the default behavior for WM_SYSCOMMAND messages, although applications can add, delete, and change system-menu entries.

Menu Items

All menus can contain two main types of menu items: command items and submenu items. When the user chooses a command item, the menu immediately posts a message to the parent window. When the user selects a submenu item, the menu displays a submenu from which the user may choose another item. Since a submenu window also can contain a submenu item, submenus can originate from other submenus.

When the user chooses a command item from a menu, the menu system posts a WM_COMMAND, WM_SYSCOMMAND, or WM_HELP message to the owner window, depending on the style bits of the menu item.

Applications can change the attributes, style, and contents of menu items, and insert and delete items at run time, to reflect changes in the command environment. An application also can add items to or delete items from the menu bar, a pop-up menu, or a submenu. For example, an application might maintain a menu of the fonts currently available in the system. This application would use graphics programming interface (GPI) calls to determine which fonts were available and, then, insert a menu item for each font into a submenu. Furthermore, the application might set the check-mark attribute of the menu item for the currently chosen font. When the user chose a new font, the application would remove the check-mark attribute from the previous choice and add it to the new choice.

The Help Item

To present a standard interface to the novice user, all applications must have a Help item in their menu bars. The Help item is defined with a particular style, attributes, and position in the menu. When the user chooses the Help item, the menu posts a WM_HELP message to the owner window, enabling the application to respond appropriately.

The item should read Help, have an identifier of 0, and have the MIS_BUTTONSEPARATOR or MIS_HELP item styles. The Help menu item should be the last item in the menu template, so that it is displayed as the rightmost item in the menu bar.

If an application uses the system default accelerator table, the user can select the Help item using either a mouse or the F1 key.

Menu-Item Styles

All menu items have a combination of style bits that determine what kind of data the item contains and what kind of message it generates when the user selects it. For example, a menu item can have the MIS_TEXT, MIS_BITMAP, or other styles that specify the visual representation of the menu item on the screen. Other styles determine what kinds of messages the item sends to its owner and whether the owner draws the item. Menu-item styles typically do not change during program execution, but you can query and set them dynamically by sending MM_QUERYITEM and MM_SETITEM messages with the menu-item identifier to the menu-bar window. For text menu items (MIS_TEXT), an MM_SETITEMTEXT message sets the text. The MM_QUERYITEMTEXT message queries the text of the item. For non-text menu items, the *hItem* field of the MENUITEM structure typically contains the handle of a display object, such as a bit-map handle for MIS_BITMAP menu items.

An application can draw a menu item by setting the style MIS_OWNERDRAW for the menu item. This usually is done by specifying the MIS_OWNERDRAW style for the menu item in the resource-definition file; but it also can be done at run time. When the application draws a

menu item, it must respond to messages from the menu each time the item must be drawn.

Menu-Item Attributes

Menu items have attributes that determine how the items are displayed and whether or not the user can choose them. An application can set and query menu-item attributes by sending MM_SETITEMATTR and MM_QUERYITEMATTR messages, with the menu-item identifier, to the menu-bar window. If the specified item is in a submenu, there are two methods of determining its attributes. The first is to send MM_SETITEMATTR and MM_QUERYITEMATTR messages to the top-level menu, specifying the identifier of the item and setting a flag so that the message searches all submenus for the item. Then, you can retrieve the handle of the menu-bar by calling WinWindowFromID, with the handle of the frame window and the FID_MENU frame-control identifier.

The second method, which is more efficient if you want to either work with more than one submenu item or set the same item several times, involves two steps:

1. Send an MM_QUERYITEM message to the menu, with the identifier of the submenu. The updated MENUITEM structure contains the window handle of the submenu.
2. Send an MM_QUERYITEMATTR (or MM_SETITEMATTR) message to the submenu window, specifying the identifier of the item in the submenu.

Menu-Item Structure

A single menu item is defined by the MENUITEM data structure. This structure is used with the MM_INSERTITEM message to insert items in a menu or to query and set item characteristics with the MM_QUERYITEM and MM_SETITEM messages. The MENUITEM structure has the following form:

```
typedef struct _MENUITEM { /* mi */
    SHORT iPosition;
    USHORT afStyle;
    USHORT afAttribute;
    USHORT id;
    HWND hwndSubMenu;
    ULONG hItem;
} MENUITEM;
```

You can derive the values of most of the fields in this structure directly from the resource-definition file. However, the last field in the structure, *hItem*, depends on the style of the menu item.

The *iPosition* field specifies the ordinal position of the item within its menu window. If the item is part of the menu bar, *iPosition* specifies its relative left-to-right position, with 0 being the leftmost item. If the item is part of a submenu, *iPosition* specifies its relative top-to-bottom and left-to-right positions, with 0 being the upper-left item. An item with the MIS_BREAKSEPARATOR style in a pull-down menu causes a new column to begin.

The *afStyle* field contains the style bits of the item. The *afAttribute* field contains the attribute bits.

The *id* field contains the menu-item identifier. The identifier *should* be unique but does not *have* to be. Just remember that, when multiple items have the same identifier, they post the same command number in the WM_COMMAND, WM_SYSCOMMAND, and WM_HELP messages. Also, any message that specifies a menu item with a non-unique identifier will find the first item that has that identifier.

The *hwndSubMenu* field contains the window handle of a submenu window (if the item is a submenu item). The *hwndSubMenu* field is NULL for command items.

The *hItem* field contains a handle to the display object for the item, unless the item has the MIS_TEXT style, in which case, *hItem* is 0. For example, a menu item with the MIS_BITMAP style has an *hItem* field that is equal to its bit-map handle.

Menu Access

The OS/2 operating system is designed to work with or without a mouse or other pointing device. The system provides default behavior that enables a user to interact with menus without a mouse. Following are the keystrokes that produce this default behavior:

Keystroke	Action
Alt	Toggles in and out of menu-bar mode.
Alt+Spacebar	Shows the system menu.
F10	Backs up one level. If a submenu is displayed, it is canceled. If no submenu is displayed, this keystroke exits the menu.
Shift+Esc	Shows the system menu.
Right Arrow	Cycles to the next top-level menu item. If the selected item is at the far-left side of the menu, the menu code sends a WM_NEXTMENU message to the frame window. The default processing by the frame window is to cycle between the application and system menus. (An application can modify this behavior by subclassing the frame window.) If the selected item is in a submenu, the next column in the submenu is selected, or the next top-level menu item is selected; this keystroke also can send or process a WM_NEXTMENU message.
Left Arrow	Works like the Right Arrow key, except in the opposite direction. In submenus, this keystroke backs up one column, except when the currently selected item is in the far-left column, in which case the previous submenu is selected.
Up Arrow or Down Arrow	When pressed in a top-level menu, activates a submenu. When pressed in a submenu, this keystroke selects the previous or next or item, respectively.
Enter	Activates a submenu, and highlights the first item if an item has a submenu associated with it; otherwise, this keystroke chooses the item as though the user released the mouse button while the item was selected.
Alphabetic character	Selects the first menu item with the specified character as its mnemonic key. A mnemonic is defined for a menu item by placing a tilde (~) before the character in the menu text. If the selected item has a submenu associated with it, the menu is displayed, and the first item is highlighted; otherwise, the item is chosen.

An application does not support the default keyboard behavior with any unusual code; instead, the application receives a message when a menu item is chosen by the keyboard just as though it had been chosen by a mouse.

Mnemonics

Adding mnemonics to menu items is one way of providing the user with keyboard access to menus. You can indicate a mnemonic keystroke for a menu item by preceding a character in the item text with a tilde, as in *~nFile*. Then, the user can choose that item by pressing the mnemonic key when the menu is active.

The menu bar is *active* when the user presses and releases the Alt key, and the first item in the menu bar is highlighted. A pop-up or pull-down menu is active when it is *open*.

Accelerators

In addition to mnemonics, a menu item can have an associated *keyboard accelerator*. Accelerators are different from mnemonics in that the menu need not be active for the accelerator key to work. If you have associated a menu item with a keyboard accelerator, display the accelerator to the right of the menu item. Do this in the resource-definition file by placing a tab character (\t) in the menu text before the characters that will be displayed on the right. For example, if the Close item had the F3 function key as its keyboard accelerator, the text for the item would be Close\tF3.

Using Menus

This section explains how to perform the following tasks:

- Define menu items in a resource file
- Include a menu bar in a standard window
- Create a pop-up menu
- Add a menu to a dialog window
- Access the system menu
- Respond to a the menu choice of a user
- Set and query menu-item attributes
- Add and delete menu items
- Create a custom menu item

Defining Menu Items in a Resource File

Typically, a menu resource represents the menu bar or pop-up menu and all the related submenus. A menu-item definition is organized as shown in the following code:

```
MENUITEM item text, item identifier, item style, item attributes
```

The menu resource-definition file specifies the text of each item in the menu, its unique identifier, its style and attributes, and whether it is a command item or a submenu item. A menu item that has no specification for style or attributes has the default style of MIS_TEXT and all attribute bits off, indicating that the item is enabled. The MIS_SEPARATOR style identifies nonselectable lines between menu items. The following figure is sample Resource Compiler source code that defines a menu resource. The code defines a menu with three submenu items in the menu bar (*File*, *Edit*, and *Font*) and a command item (*Help*). Each submenu has several command items, and the *Font* submenu has two other submenus within it.


```

MENU ID_MENU_RESOURCE
BEGIN
    SUBMENU "~File", IDM_FILE
    BEGIN
        MENUITEM "~Open...", IDM_FI_OPEN
        MENUITEM "~Close\tF3", IDM_FI_CLOSE, 0, MIA_DISABLED
        MENUITEM "~Quit", IDM_FI_QUIT
        MENUITEM "", IDM_FI_SEP1, MIS_SEPARATOR
        MENUITEM "~About Sample", IDM_FI_ABOUT
    END
    SUBMENU "~Edit", IDM_EDIT
    BEGIN
        MENUITEM "~Undo", IDM_ED_UNDO, 0, MIA_DISABLED
        MENUITEM "~", IDM_ED_SEP1, MIS_SEPARATOR
        MENUITEM "~Cut", IDM_ED_CUT
        MENUITEM "C~opy", IDM_ED_COPY
        MENUITEM "~Paste", IDM_ED_PASTE
        MENUITEM "C~lear", IDM_ED_CLEAR
    END
    SUBMENU "Font", IDM_FONT
    BEGIN
        SUBMENU "Style", IDM_FONT_STYLE
        BEGIN
            MENUITEM "Plain", IDM_FONT_STYLE_PLAIN
            MENUITEM "Bold", IDM_FONT_STYLE_BOLD
            MENUITEM "Italic", IDM_FONT_STYLE_ITALIC
        END
        SUBMENU "Size", IDM_FONT_SIZE
        BEGIN
            MENUITEM "10", IDM_FONT_SIZE_10
            MENUITEM "12", IDM_FONT_SIZE_12
            MENUITEM "14", IDM_FONT_SIZE_14
        END
    END
    MENUITEM "F1=Help", 0x00, MIS_TEXT | MIS_BUTTONSEPARATOR | MIS_HELP
END

```

To define a menu item with the MIS_BITMAP style, an application must use a tool such as Icon Editor to create a bit map, include the bit map in its resource-definition file, and define a menu in the file (as shown in the following figure). The text for the bit map menu items is an ASCII representation of the resource identifier of the bit map resource to be displayed for that item.

```

/* Bring externally created bit maps into the resource file. */
BITMAP 101 button.bmp
BITMAP 102 hirest.bmp
BITMAP 103 hizoom.bmp
BITMAP 104 hired.bmp

/* Connect a menu item with a bit map. */
SUBMENU "~Bitmaps", IDM_BITMAP
BEGIN
    MENUITEM "#101", IDM_BM_01, MIS_BITMAP
    MENUITEM "#102", IDM_BM_02, MIS_BITMAP
    MENUITEM "#103", IDM_BM_03, MIS_BITMAP
    MENUITEM "#104", IDM_BM_04, MIS_BITMAP
END

```

Including a Menu Bar in a Standard Window

If you have defined a menu resource in a resource-definition file, you can use the menu resource to create a menu bar in a standard window. You include the menu bar by using the FCF_MENU attribute flag and specifying the menu-resource identifier in a call to WinCreateStdWindow, as shown in the following code fragment:

```
#define ID_MENU_RESOURCE 100
```

```

HWND hwndFrame;
CHAR szClassName[]="MyClass";
CHAR szTitle[]="My Title";

ULONG flControlStyle = FCF_MENU      | FCF_SIZEBORDER |
                      FCF_TITLEBAR  | FCF_ACCELTABLE;

hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
    WS_VISIBLE,
    &flControlStyle,
    szClassName,
    szTitle,
    0, (HMODULE) NULL,
    ID_MENU_RESOURCE,
    NULL);

```

After you make this call, the operating system automatically includes the menu in the window, drawing the menu bar across the top of the window. When the user chooses an item from the menu, the menu posts the message to the frame window. The frame window passes any WM_COMMAND messages to the client window. (The frame window does not pass WM_SYSCOMMAND messages to the client window.) WM_HELP messages are posted to the focus window. The WinDefWindowProc function passes WM_HELP messages to the parent window. If a WM_HELP message is passed to a frame window, the frame window calls the HK_HELP hook. Your client window procedure must process these messages to respond to the user's actions.

Creating a Pop-up Menu

The following code fragment shows how to make a pop-up menu appear when the user double-clicks mouse button 2 anywhere in the parent window. The menu is positioned with the mouse pointer located on the item having the IDM_OPEN identifier and is constrained horizontally and vertically. Then, the user can select an item from the pop-up menu using mouse button 2.

```

#define ID_MENU_RESOURCE  110
#define IDM_OPEN          120

HWND hwndFrame;

MRESULT ClientWndProc(
    HWND hwnd,
    ULONG msg,
    MPARAM mp1,
    MPARAM mp2)
{
    HWND hwndMenu;
    BOOL fSuccess;

    switch (msg) {
        .
        . /* Process other messages. */
        .
        case WM_BUTTON2DBLCLK:
            hwndMenu = WinLoadMenu(hwnd, (HMODULE) NULL, ID_MENU_RESOURCE);
            fSuccess = WinPopupMenu(hwnd,
                                hwndFrame,
                                hwndMenu,
                                20,
                                50,
                                IDM_OPEN,
                                PU_POSITIONONITEM |
                                PU_HCONSTRAIN |
                                PU_VCONSTRAIN |
                                PU_MOUSEBUTTON2DOWN |
                                PU_MOUSEBUTTON2);
            .
            .
            .
    }
}

```

Adding a Menu to a Dialog Window

You might want to use menus in windows that were not created using the `WinCreateStdWindow` function. For these windows, you can load a menu resource by using the `WinLoadMenu` function and specifying the parent window for the menu. `WinLoadMenu` assigns the specified menu resource to the parent. To see the menu in the window, you must send a `WM_UPDATEFRAME` message to the parent after loading the menu resource. This strategy is especially useful for adding menus to a window created as a dialog window, but it can be used no matter what type of window is specified as the parent.

Accessing the System Menu

Although most applications do not alter the system menu, you can obtain the handle of the system menu by calling `WinWindowFromID` with a frame-window handle (or dialog-window handle) and the identifier `FID_SYSMENU`. Once you have the handle of the system menu, you can access the individual menu items by using predefined constants. For example, the following code fragment shows how to disable the *C*lose menu item in the system menu of a window:

```
HWND hwndSysMenu;
HWND hwndFrame;

hwndSysMenu = WinWindowFromID(hwndFrame, FID_SYSMENU);

WinSendMsg(hwndSysMenu, MM_SETITEMATTR,
    MPFROM2SHORT(SC_CLOSE, TRUE),
    MPFROM2SHORT(MIA_DISABLED, MIA_DISABLED));
```

Responding to a User's Menu Choice

When a user chooses a menu item, the client window procedure receives a `WM_COMMAND` message with `SHORT1FROMMP(mp1)` equal to the menu identifier of the chosen item. Your application must use the menu identifier to guide its response to the choice. Typically, the code in the client window procedure resembles the following code fragment:

```
case WM_COMMAND:
    DoMenuCommand(hwnd, SHORT1FROMMP(mp1));
    return 0;
```

The function that translates the menu identifier into an action typically resembles the following code fragment:

```
VOID DoMenuCommand(
    HWND hwnd,
    USHORT usItemID)
{
    /* Test the menu item. */
    switch (usItemID) {
        case IDM_FI_NEW:
            DoNew(hwnd);
            break;

        .
        . /* etc. */
        .
```

```

    }
}

```

The menu window sends a WM_MENUSELECT message every time the menu selection changes. SHORT1FROMMP(mp1) contains the identifier of the item that is changing state, and SHORT2FROMMP(mp2) is a 16-bit Boolean value that describes whether or not the item is chosen; the *mp2* parameter contains the handle of the menu.

If the Boolean value is FALSE, the item is selected but not chosen; for example, the user may have moved the cursor or mouse pointer over the item while the button was down. An application can use this message to display Help information at the bottom of the application window. The return value is ignored.

If the Boolean value is TRUE, the item is chosen—that is, the user pressed Enter or released the mouse button while an item was selected. If the application returns FALSE, the menu does not generate a WM_COMMAND, WM_SYSCOMMAND, or WM_HELP message, and the menu is not dismissed.

Setting and Querying Menu-Item Attributes

Menu-item attributes are represented in the *Attribute* field of the MENUITEM data structure. Typically, attributes are set in the resource-definition file of the menu and are changed at run time as required. Applications can use the MM_SETITEMATTR and MM_QUERYITEMATTR messages to set and query attributes for a particular menu item. One of the most common uses of these messages is to check and uncheck menu items to let the user know what option is selected currently. For example, if you have a menu item that should toggle between checked and unchecked each time the user selects it, you can use the following figure to change the checked attribute. In this example, you send an MM_QUERYITEMATTR message to the menu item to obtain its current checked attribute; then, you use the exclusive OR operator to toggle the state; and finally, you send the new attribute state back to the item using an MM_SETITEMATTR message.

```

usAttrib = SHORT1FROMMP(
    WinSendMsg(hwndMenu,          /* Submenu window */
    MM_QUERYITEMATTR,             /* Message */
    (MPARAM)itemID,              /* Item identifier */
    (MPARAM)MIA_CHECKED          /* Attribute mask */
);

usAttrib = MIA_CHECKED;          /* XOR to toggle checked attribute */

WinSendMsg(hwndMenu,             /* Submenu window */
    MM_SETITEMATTR,              /* Message */
    (MPARAM)itemID,              /* Item identifier */
    MPFROM2SHORT(MIA_CHECKED, usAttrib)); /* Attribute mask, value */

```

Adding and Deleting Menu Items

An application can add and delete items from its menus dynamically by sending MM_INSERTITEM and MM_DELETEITEM messages to the menu window. Any item, including those in submenus, can be deleted by sending a message to the menu window. Messages to insert items in submenus must be sent to the submenu's window (rather than to the window of the top-level menu). You can retrieve the handle of a submenu of the menu bar by sending an MM_QUERYITEM message to the menu-bar and specifying the identifier of the submenu item for the submenu, as shown in the following code fragment:

```

/* IDM_MYMENUID is the identifier of the submenu containing the item. */

MENUITEM mi;
HWND hwndMenu, hwndSubMenu, hwndPullDown, hwndFrame;

hwndMenu = WinWindowFromID(hwndFrame, FID_MENU);
WinSendMsg(hwndMenu,                      /* Handle of menu bar */

```

```

        MM_QUERYITEM,                /* Message */
        MPFROM2SHORT(IDM_MYMENUID, TRUE), /* Submenu identifier */
        (MPARAM) &mi);              /* Pointer to MENUITEM */

hwndPullDown = mi.hwndSubMenu;      /* Handle to submenu */

```

Once the application has the handle of the submenu, it can insert an item by filling in a MENUITEM structure and sending an MM_INSERTITEM message to the submenu. For text-menu items, the application must send a pointer to the text string as well as to the MENUITEM structure, as shown in the following figure.

```

PSZ pszNewItemString;

mi.iPosition = MIT_END;
mi.afStyle = MIS_TEXT;
mi.afAttribute = 0;
mi.id = IDM_MYMENU_FIRST;
mi.hwndSubMenu = NULL;
mi.hItem = 0;

WinSendMsg(hwndPullDown, MM_INSERTITEM, (MPARAM) &mi,
            (MPARAM) pszNewItemString);

```

To delete an item, the application sends an MM_DELETEITEM message to the menu bar, specifying the identifier of the item to delete. For example, to clear all the items following IDM_MYMENU_FIRST in a submenu in which the items are numbered sequentially, use the following code:

```

USHORT usItemNum;

/* Clear all the items in MYMENU. */
hwndMenu = WinWindowFromID(hwndFrame, FID_MENU);
usItemNum = IDM_MYMENU_FIRST;
while (WinSendMsg(hwndMenu, MM_DELETEITEM,
        MPFROM2SHORT(usItemNum++, TRUE), NULL) != 0);

```

Adding a complete submenu to the menu bar is a more complicated procedure than that shown in the previous examples. There are two strategies. The recommended technique is to define all possible submenus in your resource-definition file; and then, as your application runs, selectively remove and insert the submenus as needed.

For example, assume that your application has a submenu that you want to be displayed only when a particular application tool is in use. You must first define the submenu as part of the main menu resource in your resource-definition file, so that the system reads in the resource menu template and creates the submenu window along with the rest of the menu. You then can remove the submenu from the menu bar, saving the title of the submenu and the MENUITEM structure that defines the submenu, as shown in the following figure:

```

HWND hwndMenu, hwndClient;
MENUITEM mi;
CHAR szMenuTitle[MAX_STRINGSIZE];

/* Remove a submenu so that you can replace it later. */

/* Obtain the handle of a menu. */
hwndMenu = WinWindowFromID(WinQueryWindow(hwndClient, QW_PARENT),
        FID_MENU);

/* Obtain information on the item to remove. */
WinSendMsg(hwndMenu, MM_QUERYITEM,
        MPFROM2SHORT(IDM_MENUID, TRUE), /* TRUE to search submenus */
        (MPARAM)&mi);

/* Save the text for the submenu item. */
WinSendMsg(hwndMenu, MM_QUERYITEMTEXT,
        MPFROM2SHORT(IDM_FONT, MAX_STRINGSIZE),
        (MPARAM)szMenuTitle);

/* Remove the item, but retain mi and szMenuTitle. */
WinSendMsg(hwndMenu, MM_REMOVEITEM,

```

```
MPFROM2SHORT(IDM_FONT, TRUE), NULL);
```

It is important to use the `MM_REMOVEITEM` message, rather than `MM_DELETEITEM`, to remove the item; deleting the item destroys the submenu window-removing it does not. The submenu should remain intact so that you can insert it later.

To reinsert the submenu, send an `MM_INSERTITEM` message to the menu bar, passing the `MENUITEM` structure and menu title that you saved when you removed the item. The following code fragment shows how to insert a submenu that was removed by using the previous code example.

```
/* Put the submenu back in and obtain the handle of the menu bar. */
hwndMenu = WinWindowFromID(
    WinQueryWindow(hwndClient, QW_PARENT), FID_MENU);

/* Use the information that you saved when you removed the menu. */
WinSendMsg(hwndMenu, MM_INSERTITEM, (MPARAM)&mi,
    (MPARAM)szMenuTitle);
```

The other technique that you can use to insert a submenu in the menu bar is to build up, in memory, a data structure as a menu template and use that template and `WinCreateWindow` to create a submenu. The resultant submenu window handle then is placed in the `hwndSubMenu` field of a `MENUITEM` structure, and the menu item is sent to the menu bar with an `MM_INSERTITEM` message.

You also can create an empty submenu window by using `WinCreateWindow`. Pass `NULL` for the `pCtlData` and `pPresParams` parameters, instead of building the menu template in memory. Then insert a new menu item in the menu bar by using the `MM_INSERTITEM` message, setting the `MIS_SUBMENU` style, and putting the window handle of the created menu into the `hwndSubMenu` field. Then use the `MM_INSERTITEM` message to insert the items in the new pull-down menu.

Creating a Custom Menu Item

Applications can customize the appearance of an individual menu item by setting the `MIS_OWNERDRAW` style bit for the item. The operating system sends two different messages to an application that include owner-drawn menu items: `WM_MEASUREITEM` and `WM_DRAWITEM`. Both messages include a pointer to an `OWNERITEM` data structure.

`WM_MEASUREITEM` is sent only once for each owner-drawn item when the menu is initialized. The message is sent to the owner of the menu (typically, a frame window), which forwards the message to its client window. Typically, the client window procedure processes `WM_MEASUREITEM` by filling in the *yTop* and *Right* fields of the `RECTL` structure, specified by the *rcItem* field of this `OWNERITEM` structure; this specifies the size of the rectangle needed to enclose the item when it is drawn. The following code fragment responds to a `WM_MEASUREITEM` message.

```
case WM_MEASUREITEM:
    ((POWNERITEM) mp2)->rcItem.xRight = 26;
    ((POWNERITEM) mp2)->rcItem.yTop = 10;
    return 0;
```

If a menu item has the `MIS_OWNERDRAW` style, the owner window receives a `WM_DRAWITEM` message every time the menu item needs to be drawn. You process this message by using the *hps* and *rcItem* fields of the `OWNERITEM` structure to draw the item. There are two situations in which the owner window receives a `WM_DRAWITEM` message:

- When the item must be redrawn completely
- When the item must be highlighted or have its highlight removed

You can choose to handle one or both of these situations. Typically, you handle the drawing of the item. You may not want to handle the second situation, however, since the system-default behavior (inverting the bits in the item rectangle) often is acceptable. The two situations in which a `WM_DRAWITEM` message is received are detected by comparing the values of the *IsState* and *IsStateOld* fields of the `OWNERITEM` structure that is sent as part of the message. If the two fields are the same, draw the item. Before drawing the item, however, check its attributes to see whether it has the attributes `MIA_CHECKED`, `MIA_FRAMED`, or `MIA_DISABLED`. Then draw the item according to the attributes.

For example, when the checked attribute of an owner-drawn menu item changes, the system sends a `WM_DRAWITEM` message to the

item so that it can redraw itself and either draw or remove the check mark. If you want the system-default check mark, simply draw the item and leave the *fsAttribute* and *fsAttributeOld* fields unchanged; the system draws the check mark if necessary. If you draw the check mark yourself, clear the MIA_CHECKED bit in both *fsAttribute* and *fsAttributeOld* so that the system does not attempt to draw a check mark.

In the same example, if *fsAttribute* and *fsAttributeOld* are not equal, the highlight showing that an item is selected needs to change. The MIA_HILITED bit of the *fsAttribute* field is set if the item needs to be highlighted and is not set if the highlight needs to be removed. If you do not want to provide your own highlighting, you should ignore any WM_DRAWITEM message in which *fsAttribute* and *fsAttributeOld* are not equal. If you do not alter these two fields, the system performs its default highlighting operation. If you want to provide your own visual cue that an item is selected, respond to a WM_DRAWITEM message in which the *fsAttribute* and *fsAttributeOld* fields are not equal by providing the cue and clearing the MIA_HILITED bit of both fields before returning from the message.

Likewise, the MIA_CHECKED and MIA_FRAMED bits of *fsAttribute* and *fsAttributeOld* either can be used to perform the corresponding action or passed on, unchanged, so that the system performs the action. The following code fragment shows how to respond to a WM_DRAWITEM message when you want to draw the item and also be responsible for its highlighted state.

```
case WM_DRAWITEM:
{
    POWNERITEM poi;
    RECTL      rcl;
    MPARAM      mp2;

    poi = (POWNERITEM) mp2;

    /*
     * If the new attribute equals the old attribute,
     * redraw the entire item.
     */

    if (poi->fsAttribute == poi->fsAttributeOld) {

        /*
         * Draw the item in poi->hps and poi->rclItem, and check the
         * attributes for check marks. If you produce your own check marks,
         * use this line of code:
         *
         *     poi->fsAttributeOld = (poi->fsAttribute &= ~MIA_CHECKED);
         */

    }

    /* Else highlight the item or remove its highlight. */

    else if ((poi->fsAttribute & MIA_HILITED) !=
             (poi->fsAttributeOld & MIA_HILITED)) {

        /*
         * Set bits the same so that the menu window does not highlight
         * the item or remove its highlight.
         */

        poi->fsAttributeOld = (poi->fsAttribute &= ~MIA_HILITED);
    }
    return TRUE; /* TRUE means the item is drawn. */
} /* endcase */
```

Responding to WM_DRAWITEM Message

Messages and Message Queues

The OS/2 operating system uses messages and message queues to communicate with applications and the windows belonging to those applications. This chapter explains how to create and use messages and message queues in PM applications.

About Messages and Message Queues

Unlike traditional applications that take complete control of the computer's keyboard, mouse, and screen, PM applications must share these resources with other applications that are running at the same time. All applications run independently and rely on the operating system to help them manage shared resources. The operating system does this by controlling the operation of each application, communicating with each application when there is keyboard or mouse input or when an application must move and size its windows.

Messages

A *message* is information, a request for information, or a request for an action to be carried out by a window in an application.

The operating system, or an application, sends or posts a message to a window so that the window can use the information or respond to the request.

There are three types of messages:

- User-initiated
- Application-initiated
- System-initiated

A user-initiated message is the direct result of a user action, such as selecting a menu item or pressing a key. An application-initiated message is generated by one window in the application to communicate with another window. System-initiated messages are generated by the interface as the indirect result of a user action (for example, resizing a window) or as the direct result of a system event (such as creating a window).

A message that requires an immediate response from a window is sent directly to the window by passing the message data as arguments to the window procedure. The window procedure carries out the request or lets the operating system carry out default processing for the message.

A message that does not require an immediate response from a window is *posted* (the message data is copied) to the application's *message queue*. The message queue is a storage area that the application creates to receive and hold its posted messages. Then, the application can retrieve a message at the appropriate time, sending it to the addressed window for processing.

Every message contains a *message identifier*, which is a 16-bit integer that indicates the purpose of the message. When a window processes a message, it uses the message identifier to determine what to do.

Every message contains a *window handle*, which identifies the window the message is for. The window handle is important because most message queues and window procedures serve more than one window. The window handle ensures that the application forwards the message to the proper window.

A message contains two *message parameters*-32-bit values that specify data or the location of data that a window uses when processing the message. The meaning and value of a message parameter depend on the message. A message parameter can contain an integer, packed bit flags, a pointer to a structure that contains additional data, and so forth. Some messages do not use message parameters and, typically, set the parameters to NULL. An application always checks the message identifier to determine how to interpret the message parameters.

A *queue message* is a QMSG data structure that contains six data items, representing the window handle, message identifier, two message parameters, message time, and mouse-pointer position. The time and position are included because most queue messages are input messages, representing keyboard or mouse input from the user. The time and position also help the application identify the context of the message. The operating system posts a queue message by filling the QMSG structure and copying it to a message queue.

A *window message* consists of the window handle, the message identifier, and two message parameters. A window message does not include the message time and mouse-pointer position, because most window messages are requests to perform a task that is not related to the current time or mouse-pointer position. The operating system sends a window message by passing these values, as individual arguments, to a window procedure.

Message Queues

Every PM application must have a *message queue*. A message queue is the only means an application has to receive input from the keyboard or mouse. *Only applications that create message queues can create windows.*

An application creates a message queue by using the WinCreateMsgQueue function. This function returns a handle that the application can

use to access the message queue. After an application creates a message queue, the system posts messages intended for windows in the application to that queue. The application can retrieve queue messages by specifying the message-queue handle in the `WinGetMsg` function. It also can examine messages, without retrieving them, by using the `WinPeekMsg` function. When an application no longer needs the message queue, it can destroy the queue by using the `WinDestroyMsgQueue` function.

One message queue serves all the windows in a thread. This means a queue can hold messages for several windows. A message specifies the handle of the window to which it belongs so the application can forward a message easily to the appropriate window. The message loop recognizes a NULL window handle and the message is processed within the message loop rather than passed to `WinDispatchMessage`. See the following figure for an example of an input-message processing loop.

An application that has more than one thread can create more than one message queue. The system allows one message queue for each thread. A message queue created by a thread belongs to that thread and has no connection to other queues in the application. When an application creates a window in a given thread, the system associates the window with the message queue in that thread. The system then posts all subsequent messages intended for that window to that queue.

Note: The recommended way to structure PM applications is to have at least two threads and two message queues. The first thread and message queue control all the user-interface windows, and the second thread and message queue control all the object windows.

Several windows can use one message queue; it is important that the message queue be large enough to hold all messages that possibly can be posted to it. An application can set the size of the message queue when it creates the queue by specifying the maximum number of messages the queue can hold. The default maximum number of messages is 10.

To minimize queue size, several types of posted messages are not actually stored in a message queue. Instead, the operating system keeps a record in the queue of the message being posted and combines any information contained in the message with information from previous messages. Timer, semaphore, and paint messages are handled this way. For example, if more than one `WM_PAINT` message is posted, the operating system combines the *update regions* for each into a single update region. Although there is no actual `WM_PAINT` message in the queue, the operating system constructs one `WM_PAINT` message with the single update region when an application uses the `WinGetMsg` function.

The operating system handles mouse and keyboard input messages differently from the way it handles other types of messages. The operating system receives all keyboard and mouse events, such as keystrokes and mouse movements, into the system message queue. The operating system converts these events into messages and posts them, one at a time, to the appropriate application message queue. The application retrieves the messages from its queue and dispatches them to the appropriate window, which processes the messages.

The operating system message queue usually is large enough to hold all input messages, even if the user types or moves the mouse very quickly. If the operating system message queue does run out of space, the system *ignores* the most recent keyboard input (usually by beeping to indicate the input is ignored) and collects mouse motions into a `WM_MOUSEMOVE` message.

Every message queue has a corresponding `MQINFO` data structure that specifies the identifiers of the process and thread that own the message queue and gives a count of the maximum number of messages the queue can receive. An application can retrieve the structure by using the `WinQueryQueueInfo` function.

A message queue also has a current status that indicates the types of messages currently in the queue. An application can retrieve the queue status by using the `WinQueryQueueStatus` function. An application also can use the `WinPeekMsg` function to examine the contents of a message queue. `WinPeekMsg` checks for a specific message or range of messages in the queue and gives the application the option of removing messages from the queue. An application *can* call the `WinQueryQueueStatus` function to determine the contents of the queue before calling the `WinPeekMsg` or `WinGetMsg` function to remove a message from the queue.

Message Handling

To handle and process messages, an application can use a *message loop* and the *window procedure*. These terms are explained in the following two sections.

Message Loops

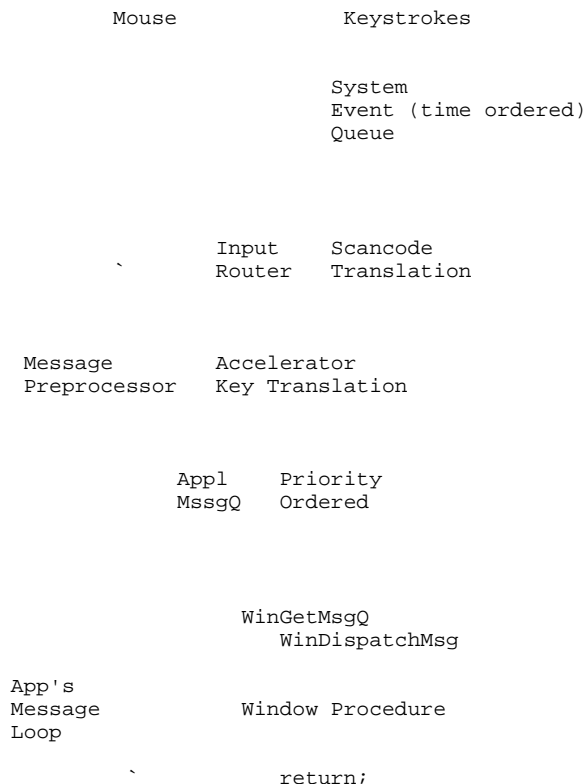
Every application with a message queue is responsible for retrieving the messages from that queue. An application can do this by using a message loop, usually in the application's main function, that retrieves messages from the message queue and dispatches them to the appropriate windows. The message loop consists of two calls: one to the `WinGetMsg` function; the other to the `WinDispatchMsg` function. The message loop has the following form:

```
HAB hab;
QMSG qmsg;

while (WinGetMsg(hab, &qmsg, NULL, 0, 0))
    WinDispatchMsg(hab, &qmsg);
```

An application starts the message loop after creating the message queue and at least one application window. Once started, the message loop continues to retrieve messages from the message queue and to dispatch (send) them to the appropriate windows. WinDispatchMsg sends each message to the window specified by the window handle in the message.

The following figure illustrates the typical routing of an input message through the operating system's and application's message loops.



Input Message Processing Loop Only one message loop is needed for a message queue, even if the queue contains messages for more than one window. Each queue message is a QMSG structure that contains the handle of the window to which the message belongs. WinDispatchMsg always dispatches the message to the proper window. WinGetMsg retrieves messages from the queue in first-in, first-out (FIFO) order, so the messages are dispatched to windows in the same order they are received.

If there are no messages in the queue, the operating system temporarily stops processing the WinGetMsg function until a message arrives. This means that processor time that, otherwise, would be spent waiting for a message can be given to the applications (or threads) that do have messages in their queues.

The message loop continues to retrieve and dispatch messages until WinGetMsg retrieves a WM_QUIT message. This message causes the function to return FALSE, terminating the loop. In most cases, terminating the message loop is the first step in terminating the application. An application can terminate its own loop by posting the WM_QUIT message in its own queue.

An application can modify its message loop in a variety of ways. For example, it can retrieve messages from the queue without dispatching them to a window. This is useful for applications that post messages without specifying a window. (These messages apply to the application rather than a specific window; they have NULL window handles.) Also, an application can direct the WinGetMsg function to search for specific messages, leaving other messages in the queue. This is useful for applications that temporarily need to bypass the usual FIFO order of the message queue.

Window Procedures

A *window procedure* is a function that receives and processes all input and requests for action sent to the windows. Every window class has a window procedure; every window created using that class uses that window procedure to respond to messages.

The system sends a message to the window procedure by passing the message data as arguments. The window procedure takes the appropriate action for the given message. Most window procedures check the message identifier, then use the information specified by the message parameters to carry out the request. When it has completed processing the message, the window procedure returns a message result. Each message has a particular set of possible return values. The window procedure must return the appropriate value for the processing it performed.

A window procedure cannot ignore a message. If it does not process a message, it must pass the message back to the operating system for default processing. The window procedure does this by calling the WinDefWindowProc function to carry out a default action and return the message result. Then, the window procedure must return this value as its own message result.

A window procedure commonly processes messages for several windows. It uses the window handle specified in the message to identify the appropriate window. Most window procedures process just a few types of messages and pass the others on to the operating system by calling WinDefWindowProc.

Posting and Sending Messages

Any application can post and send messages. Like the operating system, an application *posts* a message by copying it to a message queue. It *sends* a message by passing the message data as arguments to a window procedure. To post and send messages, an application uses the WinPostMsg and WinSendMsg functions.

An application posts a message to notify a specific window to perform a task. The WinPostMsg function creates a QMSG structure for the message and copies the message to the message queue corresponding to the given window. The application's message loop eventually retrieves the message and dispatches it to the appropriate window procedure. For example, one message commonly posted is WM_QUIT. This message terminates the application by terminating the message loop.

An application sends a message to cause a specific window procedure to carry out a task immediately. The WinSendMsg function passes the message to the window procedure corresponding to the given window. The function waits until the window procedure completes processing and then returns the message result. Parent and child windows often communicate by sending messages to each other. For example, a parent window that has an entry-field control as its child window can set the text of the control by sending a message to the child window. The control can notify the parent window of changes to the text (carried out by the user) by sending messages back to the parent window.

Occasionally, an application might need to send or post a message to all windows in the system. For example, if the application changes a system value, it must notify all windows about the change by sending a WM_SYSVALUECHANGED message. An application can send or post messages to any number of windows by using the WinBroadcastMsg function. The options in WinBroadcastMsg determine whether the message is sent or posted and specify the windows that will receive the message.

Any thread in the application can post a message to a message queue, even if the thread has no message queue of its own. However, only a thread that has a message queue can send a message. Sending a message between threads is relatively uncommon. For one reason, sending a message is costly in terms of system performance. If an application posts a message between threads, it is likely to be a semaphore message, which permits window procedures to manage a shared resource jointly.

An application can post a message without specifying a window. If the application supplies a NULL window handle when it calls the WinPostMsg function, the function posts the message to the queue associated with the current thread. The application must process the message in the message loop. This is one way to create a message that applies to the entire application instead of to a specific window.

A window procedure can determine whether it is processing a message sent by another thread by using the WinInSendMessage function. This is useful when message processing depends on the origin of the message.

A common programming error is to assume that the WinPostMsg function always succeeds. It fails when the message queue is full. An application should check the return value of the WinPostMsg function to see whether the message was posted. In general, if an application intends to post many messages to the queue, it should set the message queue to an appropriate size when it creates the queue. The default message-queue size is 10 messages.

Message Types

This section describes the three types of OS/2 messages:

- System-defined
- Application-defined
- Semaphore

System-Defined Messages

There are many *system-defined* messages that are used to control the operations of applications and to provide input and other information for applications to process. The system sends or posts a system-defined message when it communicates with an application. An application also can send or post system-defined messages. Usually, applications use these messages to control the operation of control windows created by using preregistered window classes.

Each system message has a unique message identifier and a corresponding symbolic constant. The symbolic constant, defined in the system header files, states the purpose of the message. For example, the WM_PAINT constant represents the paint message, which requests that a window paint its contents.

The symbolic constants also specify the *message category*. System-defined messages can belong to several categories; the prefix identifies the type of window that can interpret and process the messages. The following table lists the prefixes and their related message categories:

Prefix	Message category
BKM_	Notebook control
BM_	Button control
CBM_	Combination-box control
CM_	Container control
EM_	Entry-field control
LM_	List-box control
MLM_	Multiple-line entry field control
MM_	Menu control
SBM_	Scroll-bar control
SLM_	Slider control
SM_	Static control
TBM_	Title-bar control
VM_	Value set control
WM_	General window

General window messages cover a wide range of information and requests, including:

- Mouse and keyboard-input
- Menu- and dialog-input
- Window creation and management
- Dynamic data exchange (DDE)

Application-Defined Messages

An application can create messages to use in its own windows. If an application does create messages, the window procedure that receives the messages must interpret them and provide the appropriate processing.

The operating system reserves the message-identifier values in the range *0x0000* through *0x0FFF* (the value of WM_USER - 1) for system-defined messages. Applications cannot use these values for their private messages.

In addition, the operating system uses certain message values higher than WM_USER. Applications should not use these message values. A partial listing of these messages is in the following figure:

From PMSTDDL.G.H:

```
#define FDM_FILTER          WM_USER+40
#define FDM_VALIDATE       WM_USER+41
#define FDM_ERROR          WM_USER+42

#define FNTM_FACENAMECHANGED WM_USER+50
#define FNTM_POINTSIZACHANGED WM_USER+51
#define FNTM_STYLECHANGED   WM_USER+52
#define FNTM_COLORCHANGED   WM_USER+53
#define FNTM_UPDATEPREVIEW  WM_USER+54
#define FNTM_FILTERLIST     WM_USER+55
```

You should scan your header files to see if other messages have been defined with values higher than WM_USER.

Aside from the message values used by the operating system, values in the range *0x1000* (the value of WM_USER) through *0xBFFF* are available for message identifiers, defined by an application, for use in that application.

Warning: It is very important that applications do not broadcast messages in the *0x1000* through *0xBFFF* range due to the risk of misinterpretation by other applications.

Values in the range *0xC000* through *0xFFFF* are reserved for message identifiers that an application defines and registers with the system atom table; these can be used in any application. Values above *0xFFFF* (*0x00010000* through *0xFFFFFFFF*) are reserved for future use; applications must not use messages in this range.

Semaphore Messages

A *semaphore message* provides a way of signaling, through the message queue, the end of an event. An application uses a semaphore message the same way it uses system semaphore functions-to coordinate events by passing signals. A semaphore message often is used in conjunction with system semaphores.

There are four semaphore messages:

```
WM_SEM1
WM_SEM2
WM_SEM3
WM_SEM4.
```

An application posts one of these messages to signal the end of a given event. The window that is waiting for the given event receives the semaphore message when the message loop retrieves and dispatches the message.

Each semaphore message includes a bit flag that an application can use to uniquely identify the 32 possible semaphores for each semaphore message. The application passes the bit flag (with the appropriate bit set) as a message parameter with the message. The window procedure that receives the message then uses the bit flag to identify the semaphore.

To save space, the system does not store semaphore messages in the message queue. Instead, it sets a record in the queue, indicating that the semaphore message has been received, and then combines the bit flag for the message with the bit flags from previous messages. When the window procedure eventually receives the message, the bit flag specifies each semaphore message posted since the last message was retrieved.

Message Priorities

The `WinGetMsg` function retrieves messages from the message queue based on message priority. `WinGetMsg` retrieves messages with higher priority first. If it finds more than one message at a particular priority level, it retrieves the oldest message first. Messages have the following priorities:

Priority	Message
1	<code>WM_SEM1</code>
2	Messages posted using <code>WinPostMsg</code>
3	Input messages from the keyboard or mouse
4	<code>WM_SEM2</code>
5	<code>WM_PAINT</code>
6	<code>WM_SEM3</code>
7	<code>WM_TIMER</code>
8	<code>WM_SEM4</code>

Message Filtering

An application can choose specific messages to retrieve from the message queue (and ignore other messages) by specifying a message filter with the `WinGetMsg` or `WinPeekMsg` functions. The message filter is a range of message identifiers (specified by a first and last identifier), a window handle, or both. The `WinGetMsg` and `WinPeekMsg` functions use the *message filter* to select the messages to retrieve from the queue. Message filtering is useful if an application needs to search ahead in the message queue for messages that have a lower priority or that arrived in the queue later than other less important messages.

Any application that filters messages must ensure that a message satisfying the message filter can be posted. For example, filtering for a `WM_CHAR` message in a window that does not receive keyboard input prevents the `WinGetMsg` function from returning. Some messages, such as `WM_COMMAND`, are generated from other messages; filtering for them also can prevent `WinGetMsg` from returning.

To filter for mouse, button, and DDE messages, an application can use the following constants:

`WM_MOUSEFIRST` and `WM_MOUSELAST`
`WM_BUTTONCLICKFIRST` and `WM_BUTTONCLICKLAST`
`WM_DDE_FIRST` and `WM_DDE_LAST`.

Using Messages

This section explains how to perform the following tasks:

- Create a message queue and message loop
- Examine the message queue
- Post and send messages between windows
- Broadcast a message to multiple windows
- Use message macros

Creating a Message Queue and Message Loop

An application needs a message queue and message loop to process messages for its windows. An application creates a message queue by using the `WinCreateMsgQueue` function. An application creates a message loop by using the `WinGetMsg` and `WinDispatchMsg` functions. The application must create and show at least one window after creating the queue but before starting the message loop. The following code fragment shows how to create a message queue and message loop:

```
MRESULT EXPENTRY ClientWndProc(HWND hwnd,ULONG msg,MPARAM mp1,MPARAM mp2);

HAB hab;

int main(VOID)
{
    HMQ hmq;
    QMSG qmsg;
    HWND hwndFrame, hwndClient;
    ULONG flFrameFlags = FCF_TITLEBAR      | FCF_SYSMENU  |
                        FCF_SIZEBORDER    | FCF_MINMAX   |
                        FCF_SHELLPOSITION | FCF_TASKLIST;

                                /* Initialize the application for
                                Presentation Manager interface. */

    hab = WinInitialize(0);

                                /* Create the application
                                message queue. */
    hmq = WinCreateMsgQueue(hab, 0);

                                /* Register the window class for your
                                client window. */
    WinRegisterClass(hab, /* Anchor block handle */
                    "MyClientClass", /* Class name */
                    (PFNWP) ClientWndProc, /* Window procedure */
                    CS_SIZEREDRAW, /* Class style */
                    0); /* Extra bytes to reserve */

                                /* Create a main window. */
    hwndFrame = WinCreateStdWindow(
        HWND_DESKTOP, /* Parent window handle */
        WS_VISIBLE, /* Style of frame window */
        &flFrameFlags, /* Frame controls */
        "MyClientClass", /* Window class for client */
        (PSZ) NULL, /* No title-bar text */
        WS_VISIBLE, /* Style of client window */
        (HMODULE) NULL, /* Module handle for resources */
        0, /* No resource identifier */
        &hwndClient); /* Pointer to client handle */

                                /* Start the message loop. */
    while (WinGetMsg(hab, &qmsg, (HWND) NULL, 0, 0))
        WinDispatchMsg(hab, &qmsg);

                                /*. Destroy the main window. */
    WinDestroyWindow(hwndFrame);

                                /* Destroy the message queue. */
    WinDestroyMsgQueue(hmq);

                                /* Terminate the application. */
    WinTerminate(hab);
}
```

Both the `WinGetMsg` and `WinDispatchMsg` functions take a pointer to a `QMSG` structure as a parameter. If a message is available, `WinGetMsg` copies it to the `QMSG` structure; `WinDispatchMsg` then uses the data in the structure as arguments for the window procedure.

Occasionally, an application might need to process a message before dispatching it. For example, if a message is posted but the destination window is not specified (that is, the message contains a `NULL` window handle), the application must process the message to determine which window should receive the message. Then the `WinDispatchMsg` function can forward the message to the proper window. The following code fragment shows how the message loop can process messages that have `NULL` window handles:

```

HAB hab;
QMSG qmsg;

while (WinGetMsg (hab, &qmsg, (HWND) NULL, 0, 0)) {
    if (qmsg.hwnd == NULL) {
        .
        . /* Process the message. */
        .
    }
    else
        WinDispatchMsg (hab, &qmsg);
}

```

Examining the Message Queue

An application can examine the contents of the message queue by using the `WinPeekMsg` or `WinQueryQueueStatus` function. It is useful to examine the queue if the application starts a lengthy operation that additional user input might affect, or if the application needs to look ahead in the queue to anticipate a response to user input.

An application can use `WinPeekMsg` to check for specific messages in the message queue. This function is useful for extracting messages for a specific window from the queue. It returns immediately if there is no message in the queue. An application can use `WinPeekMsg` in a loop without requiring the loop to wait for a message to arrive. The following code fragment checks the queue for `WM_CHAR` messages:

```

HAB hab;
QMSG qmsg;

if (WinPeekMsg(hab, &qmsg, (HWND) NULL, WM_CHAR, WM_CHAR, PM_NOREMOVE)){
    .
    . /* Process the message. */
    .
}

```

An application also can use the `WinQueryQueueStatus` function to check for messages in the queue. This function is very fast and returns information about the kinds of messages available in the queue and which messages have been posted recently. Most applications use this function in message loops that need to be as fast as possible.

Posting a Message to a Window

An application can use the `WinPostMsg` function to post a message to a window. The message goes to the window's message queue. The following code fragment posts the `WM_QUIT` message.

```

HWND hwnd;

if (!WinPostMsg(hwnd, WM_QUIT, NULL, NULL)){
    .
    . /* Message was not posted. */
    .
}

```

The `WinPostMsg` function returns `FALSE` if the queue is full, and the message cannot be posted.

Sending a Message to a Window

An application can use the `WinSendMessage` function to send a message directly to a window. An application uses this function to send messages to child windows. For example, the following code fragment sends an `LM_INSERTITEM` message to direct a list-box control to add an item to the end of its list:

```
HWND hwndListBox;
static CHAR szWeekday[] = "Tuesday";

WinSendMessage(hwndListBox,
               LM_INSERTITEM,
               (MPARAM)LIT_END,
               MPFROMP(szWeekday));
```

`WinSendMessage` calls the window's window procedure and waits for it to handle the message and return a result. An application can send a message to any window in the system, as long as the application has the handle of the target window. The message queue does not store the message; however, the thread making the call must have a message queue.

Broadcasting a Message

An application can send a message to multiple windows by using the `WinBroadcastMessage` function. Often this function is used to broadcast the `WM_SYSVALUECHANGED` message after an application changes a system value. The following code fragment shows how to broadcast this message to all frame windows in all applications:

```
HWND hwnd;

WinBroadcastMessage(
    hwnd,                      /* Window handle */
    WM_SYSVALUECHANGED,       /* Message identifier */
    NULL,                      /* No message parameters */
    NULL,                      /* No message parameters */
    BMSG_FRAMEONLY | BMSG_POSTQUEUE); /* All frame windows */
```

An application can broadcast messages to all windows, just frame windows, or just the windows in the application.

Using Message Macros

The system header files define several macros that help create and interpret message parameters.

One set of macros helps you construct message parameters. These macros are useful for sending and posting messages. For example, the following code fragment uses the `MPFROMSHORT` macro to convert a 16-bit integer into the 32-bit message parameter:

```
HWND hwndButton;

WinSendMessage(hwndButton, BM_SETCHECK, MPFROMSHORT(1), NULL);
```

A second set of macros helps you extract values from a message parameter. These macros are useful for handling messages in a window procedure. The following code fragment determines whether the window receiving the WM_FOCUSCHANGE message is gaining or losing the keyboard focus. The fragment uses the SHORT1FROMMP macro to extract the focus-change flag, the SHORT2FROMMP macro to extract the focus flag, and the HWNDFROMMP macro to extract the window handle.

```
USHORT fsFocusChange;
MPARAM mp1, mp2;
HWND hwndGainFocus;

case WM_FOCUSCHANGE:
    fsFocusChange = SHORT2FROMMP(mp2);    /* Gets focus-change flags */
    if (SHORT1FROMMP(mp2))                /* Gaining or losing focus? */
        hwndGainFocus = HWNDFROMMP(mp1);
```

A third set of macros helps you construct a message result. These macros are useful for returning message results in a window procedure, as the following code fragment illustrates:

```
return (MRFROM2SHORT(1, 2));
```

Multiple-Line Entry Field Controls

A *multiple-line entry (MLE) field* is a sophisticated control window that enables a user to view and edit multiple lines of text. This chapter describes how to create and use multiple-line entry field controls in PM applications.

About Multiple-Line Entry Field Controls

An MLE field control gives an application the text-editing capabilities of a simple text editor. The application can create a multiple-line entry field by using WinCreateWindow or by specifying the MLE statement in a dialog-window template in a resource-definition file.

MLE Styles

The style of an MLE field control determines how the MLE field appears and behaves. An application can specify a combination of the following styles for an MLE field:

Style Name	Description
MLS_BORDER	Draws a border around the MLE field.
MLS_DISABLEUNDO	Directs the MLE control not to allow undo actions.
MLS_HSCROLL	Adds a horizontal scroll bar to the MLE field. The MLE control enables this scroll bar whenever any line exceeds the width of the MLE field.
MLS_IGNORETAB	Directs the MLE control to ignore the Tab key. It passes the appropriate WM_CHAR to its owner window.

MLS_LIMITVSCROLL	Displays the last MLE line at the bottom of the screen page. When this style is not used, the MLE control shows an empty space between the last MLE line and the bottom of the screen page.
MLS_READONLY	Prevents the MLE field from accepting text from the user. This style is useful for displaying lengthy static text in a client or dialog window.
MLS_VSCROLL	Adds a vertical scroll bar to the MLE field. The MLE control enables this scroll bar whenever the number of lines exceeds the height of the MLE field.
MLS_WORDWRAP	Automatically breaks lines that are longer than the width of the MLE field.

MLE Notification Codes

An MLE field control sends WM_CONTROL messages containing notification codes to its owner whenever certain events occur, for example, when the user or application tries to insert too much text, or when the user uses the scroll bars. The owner window uses the notification codes either to carry out custom operations for the MLE field or to respond to errors.

The MLE field control sends the MLN_HSCROLL or MLN_VSCROLL notification codes when the user enables the scroll bars so that the application can monitor the visible contents of the MLE field. The application also can monitor the contents of an MLE field by using the MLM_QUERYFIRSTCHAR message, which specifies the offset of the character in the upper-left corner of the MLE field. This represents the first MLE character that is visible to the user. To provide an alternative way of scrolling the contents of an MLE field, an application can move the character at the specified offset to the upper-left corner of an MLE field using the MLM_SETFIRSTCHAR message.

The MLE field control sends an MLN_CHANGE notification code when the user changes the text in some way. This notification code is especially useful when the MLE field is in a dialog window, because the dialog procedure can use this code to determine whether it should process the contents of the MLE field. If an application does not process MLN_CHANGE notification codes, it can use the MLM_QUERYCHANGED message to determine whether the user has made changes to the MLE text. The MLM_SETCHANGED message makes the MLE field control send an MLN_CHANGE notification code with every event that occurs in the MLE field, regardless of whether the user has changed anything. This code also can be used to hide a change made by a user.

MLE Text Editing

An MLE field contains one or more lines of text. Each line consists of one or more characters and ends with one or more characters that represent the end of the line. The end-of-line characters are determined by the format of the text.

The user can type text in an MLE field when the MLE field has the focus. The application can insert text at any time by using the MLM_INSERT message and specifying the text as a null-terminated string. The MLE field control inserts the text at the cursor position or replaces the selected text.

The MLE field control entry mode, insert or overstrike, determines what happens when the user inserts text. The user sets the entry mode by pressing the Insert key. The entry mode alternates each time the user presses Insert. When overstrike mode is enabled, at least one character is selected. This means that the MLM_INSERT message always replaces at least one character. If insert mode is enabled, the MLM_INSERT message replaces only those characters the user or application has selected. Otherwise, the MLE field makes room for the inserted characters by moving existing characters to the right, starting at the cursor position.

The cursor position, identified by a blinking bar, is specified as a character offset relative to the beginning of the text. The user can set the cursor position by using the mouse or Arrow keys to move the blinking bar. An application can set the cursor position by using the MLM_SETSEL message, which directs the MLE field control to move the blinking bar to a given character position. The MLM_SETSEL message also can set the selection.

The *selection* is one or more characters of text on which the MLE field control carries out an operation, such as deleting or copying. The user selects text by pressing the Shift key while moving the cursor or by pressing mouse button 1 while moving the mouse. The user also can select a word in a block of text by double-clicking on the word. An application selects text by using the MLM_SETSEL message to

specify the cursor position and the anchor point. The selection is all the text between the cursor position and the anchor point. If the cursor position and anchor point are equal, there is no selection. An application can retrieve the cursor position, anchor point, or both, by using the MLM_QUERYSEL message.

The user can delete characters, one at a time, by pressing the Delete key or the Backspace key. Pressing the Delete key deletes the character to the right of the cursor; pressing the Backspace key deletes the character to the left of the cursor and changes the cursor position. An application can delete one or more characters by using the MLM_DELETE message, which directs the MLE field control to delete a specified number of characters, starting at the given position. This message does not change the cursor position. An application can delete selected text by using the MLM_CLEAR message.

An application can reverse the previous operation by using the MLM_UNDO message, which restores the MLE field to its previous state. This is a quick way to fix editing mistakes. However, not all operations can be undone.

The application determines whether the previous operation can be undone by using the MLM_QUERYUNDO message, which returns TRUE and indicates the type of operation that can be undone. Using the MLM_RESETUNDO message, an application can prevent a subsequent MLM_UNDO message from changing the state of an MLE field.

MLE Text Formatting

An application can retrieve the number of lines of text in an MLE field by using the MLM_QUERYLINECOUNT message and can retrieve the number of characters in the MLE field by using the MLM_QUERYTEXTLENGTH message. The amount of text and, subsequently, the number of lines to be entered in an MLE field depend on the text limit. An application sets the text limit by using the MLM_SETTEXTLIMIT message and determines the current limit by using the MLM_QUERYTEXTLIMIT message. The user cannot set the text limit. If the user types to the text limit, the MLE field control beeps and ignores any subsequent keystrokes. If the application attempts to add text beyond the limit, the MLE field control truncates the text.

An application can control the length of each line in an MLE field by enabling word wrapping. When word wrapping is enabled, the MLE field control automatically breaks any line that is longer than the width of the MLE field. An application can set word wrapping by using the MLM_SETWRAP message, and it can determine whether the MLE field control is wrapping text by using the MLM_QUERYWRAP message. Word wrapping is disabled by default unless the application specifies the MLS_WORDWRAP style when creating the MLE field control.

An application can set tab stops for an MLE control by using the MLM_SETTABSTOP message. Tab stops specify the maximum width of a tab character. When the user or an application inserts a tab character, the MLE field control expands the character so that it fills the space between the cursor position and the next tab stop. The MLM_SETTABSTOP message sets the distance (in pels) between tab stops, and the MLE field control provides as many tab stops as necessary, no matter how long the line gets. An application can retrieve the distance between tab stops using the MLM_QUERYTABSTOP message.

An application can use the MLM_SETFORMATRECT message to set the *format rectangle* (MLE field). The format rectangle is used to set the horizontal and vertical limits for text. The MLE control sends a notification message to the parent window of the MLE field if text exceeds either of those limits. An application typically uses the format rectangle to provide its own word wrapping or other special text processing. An application can retrieve the current format rectangle by using the MLM_QUERYFORMATRECT message.

An application can prevent the user's editing of the MLE field by setting the MLS_READONLY style in WinCreateWindow or in the MLE statement in the resource-definition file. The application also can set and query the read-only state by using the MLM_SETREADONLY and MLM_QUERYREADONLY messages, respectively.

An application can set the colors and font for an MLE field by using the MLM_SETTEXTCOLOR, MLM_SETBACKCOLOR, and MLM_SETFONT messages. These messages affect all text in the MLE field. An MLE field cannot contain a mixture of fonts and colors. An application can retrieve the current values for the colors and font by using the MLM_QUERYTEXTCOLOR, MLM_QUERYBACKCOLOR, and MLM_QUERYFONT messages.

To prevent scrolling within the MLE when the MLS_READONLY style bit is set, use the MLM_DISABLEREFRESH message. The keyboard and mouse input can be enabled using the MLM_ENABLEREFRESH message.

MLE Text Import and Export Operations

An application can copy text to and from an MLE field by importing and exporting. To import text to an MLE field, an application can use the MLM_IMPORT message, which copies text from a buffer to the MLE field. To export text from an MLE field, the application can use the MLM_EXPORT message, which copies text from the MLE field to a buffer. The application uses the MLM_SETIMPORTEXPORT message to set the import and export buffers.

An application can import and export text in a variety of formats. A text format, set with the MLM_FORMAT message, identifies which

characters are used for the end-of-line characters. An MLE field can have the following text formats:

Format Name	Description
MLFIE_CFTEXT	Exported lines end with a carriage return/newline character pair (0x0D, 0x0A). Imported lines must end with a newline character, carriage return/newline character pair, or newline/carriage return character pair.
MLFIE_NOTRANS	Imported and exported lines end with a newline character (0x0A).
MLFIE_WINFMT	For exported lines, the carriage return/newline character pair marks a <i>hard</i> linebreak (a break entered by the user). Two carriage-return characters and a newline character (0x0D, 0x0D, 0x0A) mark a <i>soft</i> linebreak (a break inserted during word wrapping and not entered by the user). For imported lines, the extra carriage-return in soft linebreak characters is ignored.

The text format can affect the number of characters in a selection. To ensure that the export buffer is large enough to hold exported text, an application can send the MLM_QUERYFORMATLINELENGTH message. The application can send the MLM_QUERYFORMATTEXTLENGTH message to determine the number of bytes in the text to be exported.

Each time an application inserts text in an MLE field, the MLE field control automatically refreshes (repaints) the display by drawing the new text. When an application copies large amounts of text to an MLE field, refreshing can be quite time-consuming, so the application should disable the refresh state. The application disables the refresh state by sending the MLM_DISABLEREFRESH message. After copying all the text, the application can restore the refresh state by sending the MLM_ENABLEREFRESH message.

MLE Cut, Copy, and Paste Operations

The user can cut, copy, and paste text in an MLE field by using the Shift+Delete, Ctrl+Insert, and Shift+Insert key combinations, respectively. An application-either by itself or in response to the user-can cut, copy, and paste text by using the MLM_CUT, MLM_COPY, and MLM_PASTE messages. The MLM_CUT and MLM_COPY messages copy the selected text to the clipboard. The MLM_CUT message also deletes the text from the MLE field; MLM_COPY does not. The MLM_PASTE message copies the text from the clipboard to the current position in the MLE field, replacing any existing text with the copied text. An application can delete the selected text without copying it to the clipboard by using the MLM_CLEAR message.

An application also can copy the selected text from an MLE field to a buffer by using the MLM_QUERYSELTEXT message. This message does not affect the contents of the clipboard.

MLE Search and Replace Operations

An application can search for a specified string within MLE field text by using the MLM_SEARCH message, which searches for the string. The MLE field control returns TRUE if the string is found. The cursor does not move to the string unless the message specifies the MLFSEARCH_SELECTMATCH option.

An application also can use the MLM_SEARCH message to replace one string with another. If the message specifies the MLFSEARCH_CHANGEALL option, the MLE field control replaces all occurrences of the search string with the replacement string. Both the search string and the replacement string must be specified in an MLM_SEARCHDATA data structure passed with the message.

MLE Colors

For version 3, or lower, of the OS/2 operating system, MLE supports indexed (solid) colors only; it does not support dithered (RGB) colors.

For versions, higher than version 3, of the OS/2 operating system, MLE supports RGB colors. Indexed colors are changed to the closest RGB color representation.

Using Multiple-Line Entry Field Controls

This section explains how to create an MLE field control by using WinCreateWindow and by specifying the MLE statement in a dialog template in a resource-definition file.

Creating an MLE

The following sample code fragment shows how to create an MLE by using WinCreateWindow:

```
#define MLE_WINDOW_ID 2

HWND hwndParent;
HWND hwndMLE;

hwndMLE = WinCreateWindow(
    hwndParent,      /* Parent window      */
    WC_MLE,          /* Window class       */
    "Test",          /* Initial text       */
    WS_VISIBLE |     /* Window style       */
    MLS_BORDER,      /* Window style       */
    100, 100,        /* x and y positions  */
    100, 100,        /* Width and height   */
    hwndParent,      /* Owner window       */
    HWND_TOP,        /* Top of z-order     */
    MLE_WINDOW_ID,   /* Identifier         */
    NULL,            /* Control data       */
    NULL);           /* Presparam          */
```

It also is common to create an MLE field control by using an MLE statement in a dialog-window template in a resource file, as shown in the following code fragment:

```
MLE    " ",
    IDD_MLETEXT,
    110, 10, 50, 100,
    WS_VISIBLE |
    MLS_BORDER |
    MLS_WORDWRAP
```

The predefined class for an MLE control is WC_MLE. If you do not specify a style for the MLE control, the default styles used are MLS_BORDER, WS_GROUP, and WS_TABSTOP.

Importing and Exporting MLE Text

Importing and exporting MLE text takes place through a buffer. An *import* operation copies text from the buffer to the MLE field; an *export*

operation copies text from the MLE to the buffer. Before an application can import or export MLE text, it must send an MLM_SETIMPORTEXPORT message to the MLE field control, specifying the address and size of the buffer.

For version 3, or lower, of the OS/2 operating system the maximum size of import/export buffer is 64K. Once the data is into the buffer, the data is manipulated (verified for carriage returns, line feeds and so forth), and is finally placed in the MLE's memory.

Importing MLE Text

To import text, an application sends the MLM_IMPORT message to the MLE field control. This message requires two parameters: *pOffset* and *cbCopy*. The *pOffset* parameter is a pointer to a variable that specifies the position in the MLE field where the text from the buffer is to be placed. The position is an *offset* from the beginning of the MLE text, that is, the number of characters from the beginning of the MLE text. If *pOffset* points to a variable that equals -1, the MLE field control places the text starting at the current cursor position. On return, this variable contains the offset to the first character beyond the imported text. The *cbCopy* parameter of the MLM_IMPORT message points to a variable that specifies the number of bytes to import.

The following criterias apply when importing MLE text:

- If the text ends by a line feed (LF), the import logic generates a blank line.
- If the text ends by a carriage return (CR), MLE prevents a line break (LB) but flags the condition.
- If the *pOffset* field points to the current cursor position (-1) and the import text contains a LF:
 - If the MLE text is imported before the text being edited, then the cursor does not move and the text being edited is shifted down to make room for the text being imported.
 - If the MLE text is imported after the text being edited, then the cursor does not move and the text being imported is inserted starting at the current cursor position.
- If the *pOffset* field points to the current cursor position (-1) and the import text does not contain a LF:
 - If the MLE text is imported before the text being edited, then the cursor does not move and the text being edited is shifted to the right to make room for the text being imported.
 - If the MLE text is imported after the text being edited, then the cursor does not move and the text being imported is inserted starting at the current cursor position.

Exporting MLE Text

Before using the MLM_EXPORT message the number of characters to export needs to be determined. The MLM_QUERYFORMATTEXTLENGTH message is used to determine the number of characters to be copied from the MLE to the buffer (including LF and CR) and to allocate the room in the buffer. MLM_EXPORT is then used to export the MLE text into the buffer.

Note: The MLM_QUERYTEXTLENGTH message does not consider the CR and LF characters as the MLM_QUERYFORMATTEXTLENGTH message does.

The following code fragment reads text from a file to a buffer, then imports the text to an MLE field:

```
HWND  hwndMle;
CHAR  szMleBuf[512];
IPT   lOffset = 0;
PSZ   pszTextFile;
HFILE hf;
ULONG cbCopied;
ULONG ulAction;
ULONG cbBytesRead;

/* Obtain a file name from the user */

/* Open the file */
```

```

DosOpen(pszTextFile,
        &hf,
        &ulAction,
        0,
        FILE_NORMAL,
        FILE_OPEN |
        FILE_CREATE,
        OPEN_ACCESS_READONLY |
        OPEN_SHARE_DENYNONE,
        NULL);

/* Zero-fill the buffer using memset, a C run-time function */
memset(szMleBuf, 0, sizeof(szMleBuf));

/* Set the MLE import-export buffer */
WinSendMsg(hwndMle,
            MLM_SETIMPORTEXP,
            MPFROMP(szMleBuf),
            MPFROMSHORT((USHORT) sizeof(szMleBuf)));

/*****
/* Read the text from the file to the buffer,
/* then import it to the MLE.
*****/

do {
    DosRead(hf,
            szMleBuf,
            sizeof(szMleBuf),
            &cbBytesRead);

    cbCopied = (ULONG) WinSendMsg(hwndMle,
                                MLM_IMPORT,
                                MPFROMP( &lOffset),
                                MPFROMP(&cbBytesRead));

    } while (cbCopied);

/* Close the file */
DosClose(hf);

```

To export MLE text, an application sends the MLM_EXPORT message to the MLE control. Like MLM_IMPORT, the MLM_EXPORT message takes the *pOffset* and *cbCopy* parameters. The *pOffset* parameter is a pointer to a variable that specifies the offset to the first character to export. A value of -1 specifies the current cursor position. On return, the variable contains the offset to the first character in the MLE field not copied to the buffer. The *cbCopy* parameter is a pointer to a variable that specifies the number of bytes to export. On return, this variable equals 0 if the number of characters actually copied does not exceed the number specified to be copied. The following code fragment shows how to export text from an MLE field, then store the text in a file:

```

HWND  hwndMle;
CHAR  szMleBuf[512];
IPT   lOffset = 0;
PSZ   pszTextFile;
HFILE hf;

ULONG cbCopied;
ULONG ulAction;
ULONG cbBytesWritten;
ULONG cbCopy;

/* Zero-fill the buffer using memset, a C run-time function */
memset(szMleBuf, 0, sizeof(szMleBuf));

/* Set the MLE import-export buffer */
WinSendMsg(hwndMle,
            MLM_SETIMPORTEXP,
            MPFROMP(szMleBuf),
            MPFROMSHORT((USHORT) sizeof(szMleBuf)));

.
.
.

/* Obtain a filename from the user */
.
.
.

```



```

/* Open the file */
DosOpen(pszTextFile,
        &hf,
        &ulAction,
        0,
        FILE_NORMAL,
        FILE_OPEN |
        FILE_CREATE,
        OPEN_ACCESS_WRITEONLY |
        OPEN_SHARE_DENYNONE,
        NULL);

/* Find out how much text is in the MLE */
cbCopy = (ULONG) WinSendMsg(hwndMle,
                             MLM_QUERYFORMATTEXTLENGTH,
                             MPFROMLONG(1Offset),
                             MPFROMLONG((-1)));

/* Copy the MLE text to the buffer */
cbCopied = (ULONG) WinSendMsg(hwndMle,
                              MLM_EXPORT,
                              MPFROMP(&lOffset),
                              MPFROMP(&cbCopy));

/* Write the contents of the buffer to the file */
DosWrite(hf,
         szMleBuf,
         sizeof(szMleBuf),
         &cbBytesWritten);

/* Close the file */
DosClose(hf);

```

Searching MLE Text

An application uses the MLM_SEARCH message and the MLE_SEARCHDATA data structure to search for strings in MLE text. The first parameter of the MLM_SEARCH message is an array of flags that specify the style of the search. The application can set the MLFSEARCH_CASESENSITIVE flag if a case-sensitive search is required. If the application sets the MLFSEARCH_SELECTMATCH flag, the MLE field control highlights a matching string and, if necessary, scrolls the string into view. An application can use the MLFSEARCH_CHANGEALL flag to replace every occurrence of the string with the string specified in the *pchReplace* member of the MLE_SEARCHDATA data structure.

The second parameter of the MLM_SEARCH message is a pointer to an MLE_SEARCHDATA data structure that contains information required to perform the search operation. This data structure includes a pointer to the string and, if the MLFSEARCH_CHANGEALL flag is set in the MLM_SEARCH message, a pointer to the replacement string. The *iptStart* and *iptStop* members specify the starting and ending positions of the search. These positions are specified as offsets from the beginning of the MLE field. A value of -1 in the *iptStart* member causes the search to start at the current cursor position. A negative value in the *iptStop* member causes the search to end at the end of the MLE field. If a matching string is found, the MLE field control returns the length of the string in the *cchFound* member.

The following code fragment uses an entry field to obtain a search string from the user, then searches an MLE field for an occurrence of the string. The search begins at the current cursor position and ends at the end of the MLE text. When the MLFSEARCH_SELECTMATCH flag is specified, the MLE field control highlights a matching string and scrolls it into view.

The following code fragment shows how to search MLE text:

```

#define IDD_SEARCHFIELD 101

HWND hwnd;
HWND hwndEntryFld;
HWND hwndMle;
MLE_SEARCHDATA mlesrch;
CHAR szSearchString[64];

/* Obtain the handle of the entry field containing the search string */
hwndEntryFld = WinWindowFromID(hwnd, IDD_SEARCHFIELD);

```

```

/* Obtain the search string from the entry field */
WinQueryWindowText(hwndEntryFld,
    sizeof(szSearchString),
    szSearchString);

/* Fill the MLE_SEARCHDATA data structure */
mlesrch.cb = sizeof(mlesrch); /* Structure size */
mlesrch.pchFind = szSearchString; /* Search string */
mlesrch.pchReplace = NULL; /* No replacement string */
mlesrch.cchFind = 0; /* Not used */
mlesrch.cchReplace = 0; /* Not used */
mlesrch.iptStart = -1; /* Start at cursor position */
mlesrch.iptStop = -1; /* Stop at end of file */

/* Start the search operation */
WinSendMessage(hwndMle,
    MLM_SEARCH,
    MPFROMLONG(MLFSEARCH_SELECTMATCH),
    MPFROMP(&mlesrch));

```

Mouse and Keyboard Input

An OS/2 Presentation Manager application can accept input from both a mouse (or other pointing device) and the keyboard. This chapter explains how these *input events* should be received and processed.

About Mouse and Keyboard Input

Only one window at a time can receive keyboard input, and only one window at a time can receive mouse input; but they do not have to be the same window. All keyboard input goes to the window with the input focus, and, normally, all mouse input goes to the window under the mouse pointer.

System Message Queue

The operating system routes all keystrokes and mouse input to the system message queue, converting these input events into messages, and posts them, one at a time, to the proper application-defined message queues. An application retrieves messages from its queue and dispatches them to the appropriate window procedures, which process the messages.

Mouse and keyboard input events in the system message queue are strictly ordered so that a new event cannot be processed until all previous events are fully processed: the system cannot determine the destination window of an input event until then. For example, if a user types a command in one window, clicks the mouse to activate another window, then types a command in the second window, the destination of the second command depends on how the application handles the mouse click. The second command would go to the second window only if that window became active as a result of the mouse click.

It is important for an application to process all messages quickly to avoid slowing user interaction with the system. A message must be responded to immediately in the current thread, but the processing it initiates should be done asynchronously in another thread that has no windows in the desktop tree.

The OS/2 operating system can display multiple windows belonging to several applications at the same time. To manage input among these windows, the system uses the concepts of *window activation* and *keyboard focus*.

Window Activation

Although the operating system can display windows from many different applications simultaneously during a PM session, the user can interact with only one application at a time—the *active* application. The other applications continue to run, but they cannot receive user input until they become active.

To enable the user to easily identify the active application, the system activates all frames in the tree between `HWND_DESKTOP` and the window with input focus. That is, the system positions the active frame window above all other top-level windows on the screen. If the active window is a standard frame window, the window's title bar and sizing border are highlighted.

The user can control which application is active by clicking on a window or by pressing the Alt+Tab or Alt+Esc key combinations. An application can set the active frame window by calling `WinSetActiveWindow`; it also can obtain the handle of the active frame window by using `WinQueryActiveWindow`.

When one window is deactivated and another activated, the system sends a `WM_ACTIVATE` message, first to the window being deactivated, then to the window being activated. The *fActive* parameter of the `WM_ACTIVATE` message is set to `FALSE` for the window being deactivated and set to `TRUE` for the window being activated. An application can use this message to track the activation state of a client window.

Keyboard Focus

The *keyboard focus* is a temporary attribute of a window; the window that has the keyboard focus receives all keyboard input until the focus changes to a different window. The system converts keyboard input events into `WM_CHAR` messages and posts them to the message queue of the window that has the keyboard focus.

An application can set the keyboard focus to a particular window by calling `WinSetFocus`. If the application does not use `WinSetFocus` to explicitly set the keyboard-focus window, the system sets the focus to the active frame window.

The following events occur when an application uses `WinSetFocus` to shift the keyboard focus from one window (the *original* window) to another (the *new* window):

1. The system sends the original window a `WM_SETFOCUS` message (with the *fFocus* parameter set to `FALSE`), indicating that that window has lost the keyboard focus.
2. The system then sends the original window a `WM_SETSELECTION` message, indicating that the window should remove the highlight from the current selection.
3. If the original (frame) window is being deactivated, the system sends it a `WM_ACTIVATE` message (with the *fActive* parameter set to `FALSE`), indicating that the window is no longer active.
4. The system then sends the new application a `WM_ACTIVATE` message (with *fActive* set to `TRUE`), indicating that the new application is now active.
5. If the new (main) window is being activated, the system sends it a `WM_ACTIVATE` message (with *fActive* set to `TRUE`), indicating that the main window is now active.
6. The system sends the new window a `WM_SETSELECTION` message, indicating that the window should highlight the current selection.
7. Finally, the system sends the new window a `WM_SETFOCUS` message (with *fFocus* set to `TRUE`), indicating that the new window has the keyboard focus.

If, while processing a `WM_SETFOCUS` message, an application calls `WinQueryActiveWindow`, that function returns the handle of the previously-active window until the application establishes a new active window. Similarly, if the application, while processing `WM_SETFOCUS`, calls `WinQueryFocus`, that function returns the handle of the previous keyboard-focus window until the application establishes a new keyboard-focus window. In other words, even though the system has sent `WM_ACTIVATE` and `WM_SETFOCUS` messages (with the *fActive* and *fFocus* parameters set to `FALSE`) to the previous windows, those windows are considered the active and focus windows until the system establishes new active and focus windows.

If the application calls `WinSetFocus` while processing a `WM_ACTIVATE` message, the system does not send a `WM_SETFOCUS` message (with *fFocus* set to `FALSE`), because no window has the focus.

A client window receives a `WM_ACTIVATE` message when its parent frame window is being activated or deactivated. The activation or deactivation message usually is followed by a `WM_SETFOCUS` message that specifies whether the client window is gaining or losing the keyboard focus. Therefore, if the client window needs to change the keyboard focus, it should do so during the `WM_SETFOCUS` message, not during the `WM_ACTIVATE` message.

Keyboard Messages

The system sends keyboard input events as WM_CHAR messages to the message queue of the keyboard-focus window. If no window has the keyboard focus, the system posts WM_CHAR messages to the message queue of the active frame window. Following are two typical situations in which an application receives WM_CHAR messages:

An application has a client window or custom control window, either of which can have the keyboard focus. If the window procedure for the client or control window does not process WM_CHAR messages, it should pass them to WinDefWindowProc, which will pass them to the owner. Dialog control windows, in particular, should pass unprocessed WM_CHAR messages to the WinDefDlgProc function, because this is how the user interface implements control processing for the Tab and Arrow keys.

An application window owns a control window whose window procedure can handle some, but not all, WM_CHAR messages. This is common in dialog windows. If the window procedure of a control in a dialog window cannot process a WM_CHAR message, the procedure can pass the message to the WinDefDlgProc function. This function sends the message to the control window's owner, which usually is a dialog frame window. The application's dialog procedure then receives the WM_CHAR message. This also is the case when an application client window owns a control window.

A WM_CHAR message can represent a key-down or key-up transition. It might contain a character code, virtual-key code, or scan code. This message also contains information about the state of the Shift, Ctrl, and Alt keys.

Each time a user presses a key, at least two WM_CHAR messages are generated: one when the key is pressed, and one when the key is released. If the user holds down the key long enough to trigger the keyboard repeat, multiple WM_CHAR key-down messages are generated. If the keyboard repeats faster than the application can retrieve the input events from its message queue, the system combines repeating character events into one WM_CHAR message and increments a count byte that indicates the number of keystrokes represented by the message. Generally, this byte is set to 1, but an application should check each WM_CHAR message to avoid missing any keystrokes.

An application can ignore the repeat count. For example, an application might ignore the repeat count on Arrow keys to prevent the cursor from skipping characters when the system is slow.

Message Flags

Applications decode WM_CHAR messages by examining individual bits in the flag word contained in the first message parameter (*mp1*) that the system passes with every WM_CHAR message. The type of flag word indicates the nature of the message. The system can set the bits in the flag word in various combinations. For example, a WM_CHAR message can have the KC_CHAR, KC_SCANCODE, and KC_SHIFT attribute bits all set at the same time. An application can use the following list of flag values to test the flag word and determine the nature of a WM_CHAR message:

Flag Name	Description
KC_ALT	Indicates that the Alt key was down when the message was generated.
KC_CHAR	Indicates that the message contains a valid character code for a key, typically an ASCII character code.
KC_COMPOSITE	In combination with the KC_CHAR flag, this flag indicates that the character code is a combination of the key that was pressed and the previous dead key. This flag is used to create characters with diacritical marks.
KC_CTRL	Indicates that the Ctrl key was down when the message was generated.
KC_DEADKEY	In combination with the KC_CHAR flag, this flag indicates that the character code represents a dead-key glyph (such as an accent). An application displays the dead-key glyph and does not advance the

cursor. Typically, the next WM_CHAR message is a KC_COMPOSITE message, containing the glyph associated with the dead key.

KC_INVALIDCHAR	Indicates that the character is not valid for the current translation tables.
KC_INVALIDCOMP	Indicates that the character code is not valid in combination with the previous dead key.
KC_KEYUP	Indicates that the message was generated when the user released the key. If this flag is clear, the message was generated when the user pressed the key. An application can use this flag to determine key-down and key-up events.
KC_LONEKEY	In combination with the KC_KEYUP flag, this flag indicates that the user pressed no other key while this key was down.
KC_PREVDOWN	In combination with the KC_VIRTUALKEY flag, this flag indicates that the virtual key was pressed previously. If this flag is clear, the virtual key was not previously pressed.
KC_SCANCODE	Indicates that the message contains a valid scan code generated by the keyboard when the user pressed the key. The system uses the scan code to identify the character code in the current code page; therefore, most applications do not need the scan code unless they cannot identify the key that the user pressed. WM_CHAR messages generated by user keyboard input generally have a valid scan code, but WM_CHAR messages posted to the queue by other applications might not contain a scan code.
KC_SHIFT	Indicates that the Shift key was down when the message was generated.
KC_TOGGLE	Toggles on and off every time the user presses a specified key. This is important for keys like NumLock, which have an on or off state.
KC_VIRTUALKEY	Indicates that the message contains a valid virtual-key code for a key. Virtual keys typically correspond to function keys. For those using hooks, when this bit is set, KC_SCANCODE should usually be set as well.

The *mp1* and *mp2* parameters of the WM_CHAR message contain information describing the nature of a keyboard input event, as follows:

- SHORT1FROMMP (*mp1*) contains the flag word.
- CHAR3FROMMP (*mp1*) contains the key-repeat count.
- CHAR4FROMMP (*mp1*) contains the scan code.
- SHORT1FROMMP (*mp2*) contains the character code.
- SHORT2FROMMP (*mp2*) contains the virtual key code.

An application window procedure should return TRUE if it processes a particular WM_CHAR message or FALSE if it does not. Typically, applications respond to key-down events and ignore key-up events.

The following sections describe the different types of WM_CHAR messages. Generally, an application decodes these messages by creating layers of conditional statements that discriminate among the different combinations of flag and code attributes that can occur in a keyboard message.

Key-Down or Key-Up Events

Typically, the first attribute that an application checks in a WM_CHAR message is the key-down or key-up event. If the KC_KEYUP bit of the flag word is set, the message is from a key-up event. If the flag is clear, the message is from a key-down event.

Repeat-Count Events

An application can check the key-repeat count of a WM_CHAR message to determine whether the message represents more than 1 keystroke. The count is greater than 1 if the keyboard is sending characters to the system queue faster than the application can retrieve them. If the system queue fills up, the system combines consecutive keyboard input events for each key into a single WM_CHAR message, with the key-repeat count set to the number of combined events.

Character Codes

The most typical use of WM_CHAR messages is to extract a character code from the message and display the character on the screen. When the KC_CHAR flag is set in the WM_CHAR message, the low word of *wp2* contains a character code based on the current code page. Generally, this value is a character code (typically, an ASCII code) for the key that was pressed.

Virtual-Key Codes

WM_CHAR messages often contain virtual-key codes that correspond to various function keys and direction keys on a typical keyboard. These keys do not correspond to any particular glyph code but are used to initiate operations. When the KC_VIRTUALKEY flag is set in the flag word of a WM_CHAR message, the high word of *wp2* contains a virtual-key code for the key.

Note: Some keys, such as the Enter key, have both a valid character code and a virtual-key code. WM_CHAR messages for these keys will contain character codes for both newline characters (ASCII 11) and virtual-key codes (VK_ENTER).

Scan Codes

A third possible value in a WM_CHAR message is the scan code of the key that was pressed. The scan code represents the value that the keyboard hardware generates when the user presses a key. An application can use the scan code to identify the physical key pressed, as opposed to the character code represented by the same key.

Accelerator-Table Entries

The system checks all incoming keyboard messages to see whether they match any existing accelerator-table entries (in either the system

message queue or the application message queue). The system first checks the accelerator table associated with the active frame window; if it does not find a match, the system uses the accelerator table associated with the message queues. If the keyboard input event corresponds to an accelerator-table entry, the system changes the WM_CHAR message to a WM_COMMAND, WM_SYSCOMMAND, or WM_HELP message, depending on the attributes of the accelerator table. If the keyboard input event does not correspond to an accelerator-table entry, the system passes the WM_CHAR message to the keyboard-focus window.

Applications should use accelerator tables to implement keyboard shortcuts rather than translate command keystrokes. For example, if an application uses the F2 key to save a document, the application should create a keyboard accelerator entry for the F2 virtual key so that, when pressed, the F2 key generates a WM_COMMAND message rather than a WM_CHAR message.

Mouse Messages

Mouse messages occur when a user presses or releases one of the mouse buttons (a click) and when the mouse moves. All mouse messages contain the x and y coordinates of the mouse-pointer *hot spot* (relative to the coordinates of the window receiving the message) at the time the event occurs. The mouse-pointer hot spot is the location in the mouse-pointer bit map that the system tracks and recognizes as the position of the mouse pointer.

If a window has the CS_HITTEST style, the system sends the window a WM_HITTEST message when the window is about to receive a mouse message. Most applications pass WM_HITTEST messages on to WinDefWindowProc by default, so disabled windows do not receive mouse messages. Windows that specifically respond to WM_HITTEST messages can change this default behavior. If the window is enabled and should receive the mouse message, the WinDefWindowProc function (using the default processing for WM_HITTEST) returns the value HT_NORMAL. If the window is disabled, WinDefWindowProc returns HT_ERROR, in which case the window does not receive the mouse message.

The default window procedure processes the WM_HITTEST message and the *ushit* parameter in the WM_MOUSEMOVE message. Therefore, unless an application needs to return special values for the WM_HITTEST message or the *ushit* parameter, it can ignore them. One possible reason for processing the WM_HITTEST message is for the application to react differently to a mouse click in a disabled window.

The contents of the mouse-message parameters (*mp1* and *mp2*) are as follows:

- SHORT1FROMMP (*mp1*) contains the x position.
- SHORT2FROMMP (*mp1*) contains the y position.
- SHORT1FROMMP (*mp2*) contains the hit-test parameter.

Capturing Mouse Input

The operating system generally posts mouse messages to the window that is under the mouse pointer at the time the system reads the mouse input events from the system message queue. An application can change this by using the WinSetCapture function to route all mouse messages to a specific window or to the message queue associated with the current thread. If mouse messages are routed to a specific window, that window receives all mouse input until either the window releases the mouse or the application specifies another capture window. If mouse messages are routed to the current message queue, the system posts each mouse message to the queue with the *hwnd* member of the QMSG structure for each message set to NULL. Because no window handle is specified, the WinDispatchMsg function in the application's main message loop cannot pass these messages to a window procedure for processing. Therefore, the application must process these messages in the main loop.

Capturing mouse input is useful if a window needs to receive all mouse input, even when the pointer moves outside the window. For example, applications commonly track the mouse-pointer position after a mouse "button down" event, following the pointer until a "button up" event is received from the system. If an application does not call WinSetCapture for a window and the user releases the mouse button, the application does not receive the button-up message. If the application sets a window to capture the mouse and tracks the mouse pointer, the application receives the button-up message even if the user moves the mouse pointer outside the window.

Some applications are designed to require a button-up message to match a button-down message. When processing a button-down message, these applications call WinSetCapture to set the capture to their own window; then, when processing a matching button-up message, they call WinSetCapture, with a NULL window handle, to release the mouse.

Button Clicks

An application window's response to a mouse click depends on whether the window is active. The first click in an inactive window should activate the window. Subsequent clicks in the active window produce an application-specific action.

A common problem for an application that processes WM_BUTTON1DOWN or similar messages is failing to activate the window or set the keyboard focus. If the window processes WM_CHAR messages, the window procedure should call WinSetFocus to make sure the window receives the keyboard focus and is activated. If the window does not process WM_CHAR messages, the application should call WinSetActiveWindow to activate the window.

Mouse Movement

The system sends WM_MOUSEMOVE messages to the window that is under the mouse pointer, or to the window that currently has captured the mouse, whenever the mouse pointer moves. This is useful for tracking the mouse pointer and changing its shape, based on its location in a window. For example, the mouse pointer changes shape when it passes over the size border of a standard frame window.

All standard control windows use WM_MOUSEMOVE messages to set the mouse-pointer shape. If an application handles WM_MOUSEMOVE messages in some situations but not others, unused messages should be passed to the WinDefWindowProc function to change the mouse-pointer shape.

Using the Mouse and Keyboard

This section explains how to perform the following tasks:

- Determine the active status of a frame window
- Check for a key-up or key-down event
- Respond to a character message
- Handle virtual-key codes
- Handle a scan code

Determining the Active Status of a Frame Window

The activated state of a window is a frame-window characteristic. The system does not provide an easy way to determine whether a client window is part of the active frame window. That is, the window handle returned by the WinQueryActiveWindow function identifies the active frame window rather than the client window owned by the frame window.

Following are two methods for determining the activated state of a frame window that owns a particular client window:

- Call WinQueryActiveWindow and compare the window handle it returns with the handle of the frame window that contains the client window, as shown in the following code fragment:

```
HWND hwndClient;  
BOOL fActivated;  
  
fActivated = (WinQueryWindow(hwndClient, QW_PARENT) ==  
             WinQueryActiveWindow(HWND_DESKTOP));
```

- Each time the frame window is activated, the client window receives a WM_ACTIVATE message with the low word of the *mp2* equal to TRUE. When the frame window is deactivated, the client window receives a WM_ACTIVATE message with a FALSE

activation indicator.

Checking for a Key-Up or Key-Down Event

The following code fragment shows how to decode a WM_CHAR message to determine whether it indicates a key-up event or a key-down event:

```
USHORT fsKeyFlags;

case WM_CHAR: {
    USHORT fsKeyFlags = SHORT1FROMMP(mp1);

    if (fsKeyFlags & KC_KEYUP) {
        . /* Perform key-up processing. */
        .
    } else {
        . /* Perform key-down processing. */
        .
    }

    return;
}
```

Responding to a Character Message

The following code fragment shows how to respond to a character message:

```
USHORT fsKeyFlags;
UCHAR uchChr1;

case WM_CHAR:
    fsKeyFlags = (USHORT) SHORT1FROMMP(mp1);

    if (fsKeyFlags & KC_CHAR) {

        /* Get the character code from mp2. */
        uchChr1 = (UCHAR) CHAR1FROMMP(mp2);
        .
        . /* Process the character. */
        .

        return TRUE;
    }
}
```

If the KC_CHAR flag is not set, the *mp2* parameter from CHAR1FROMMP still might contain useful information. If either the Alt key or the Ctrl key, or both, are down, the KC_CHAR bit is not set when the user presses another key. For example, if the user presses the **a** key when the Alt key is down, the low word of *mp2* contains the ASCII value for "a" (0x0061), the KC_ALT flag is set, and the KC_CHAR flag is clear. If the translation does not generate any valid characters, the *char* field is set to 0.

Handling Virtual-Key Codes

The following code fragment shows how to decode a WM_CHAR message containing a valid virtual-key code:

```
USHORT fsKeyFlags;

case WM_CHAR:
fsKeyFlags = (USHORT) SHORT1FROMMP(mp1);

if (fsKeyFlags & KC_VIRTUALKEY) {

    /* Get the virtual key from mp2.          */
    switch (SHORT2FROMMP(mp2)) {
    case VK_TAB:
        . /* Process the TAB key.          */
        .
        return TRUE;
    case VK_LEFT:
        . /* Process the LEFT key.          */
        .
        return TRUE;
    case VK_UP:
        . /* Process the UP key.            */
        .
        return TRUE;
    case VK_RIGHT:
        . /* Process the RIGHT key.          */
        .
        return TRUE;
    case VK_DOWN:
        . /* Process the DOWN key.          */
        .
        return TRUE;
        . /* Etc...                          */
        .
    default:
        return FALSE;
    }
}
```

Handling a Scan Code

All WM_CHAR messages generated by keyboard input events have valid scan codes. WM_CHAR messages posted by other applications might or might not have valid scan codes. The following code fragment shows how to extract a scan code from a WM_CHAR message:

```
USHORT fsKeyFlags;
UCHAR  uchScanCode;

case WM_CHAR:
fsKeyFlags = (USHORT) SHORT1FROMMP(mp1);

if (fsKeyFlags & KC_SCANCODE) {

    /* Get the scan code from mp1.          */
    uchScanCode = CHAR4FROMMP(mp1);
    .
    . /* Process the scan code.            */
    .
```

```

    .
    return (MRESULT) TRUE;
}

```

Mouse Pointers and Icons

A *mouse pointer* is a special bit map the operating system uses to show a user the current location of the mouse on the screen. When the user moves the mouse, the mouse pointer moves on the screen. This chapter describes how to create and use mouse pointers and icons in PM applications.

About Mouse Pointers and Icons

Mouse pointers and icons are made up of bit maps that the operating system uses to paint images of the pointers or icons on the screen. A *monochrome bit map* is a series of bytes. Each bit corresponds to a single pel in the image. (The bit map representing the display typically has four bits for each pel.)

A mouse pointer or icon bit map always is twice as tall as it is wide. The top half of the bit map is an *AND* mask, in which the bits are combined, using the AND operator, with the screen bits where the pointer is being drawn. The lower half of the bit map is an *XOR* mask, in which the bits are combined, using the XOR operator, with the destination screen bits.

The combination of the AND and XOR masks results in four possible colors in the bit map. The pels of an icon or pointer can be black, white, transparent (the screen color beneath the pel), or inverted (inverting the screen color beneath the pel). The following figure shows the relationship of the bit values in the AND and XOR masks:

AND mask	0	0	1	1
XOR mask	0	1	0	1
Result	Black	White	Transparent	Inverted

Mouse-Pointer Hot Spot

Each mouse pointer has its own *hot spot*, which is the point that represents the exact location of the mouse pointer. This location is defined as an *x* and *y* offset from the lower-left corner of the mouse-pointer bit map. For the arrow-shaped pointer, the hot spot is at the tip of the arrow. For the I-beam pointer, the hot spot is at the middle of the vertical line.

Predefined Mouse Pointers

Before an application can use a mouse pointer, it first must receive a handle to the pointer. Most applications load mouse pointers from the system or from their own resource file. The operating system maintains many predefined mouse pointers that an application can use by calling WinQuerySysPointer. System mouse pointers include all the standard mouse-pointer shapes and message-box icons. The following predefined mouse pointers are available:

Mouse Pointer	Description
---------------	-------------

SPTR_APPICON	Square icon; used to represent a minimized application window.
SPTR_ARROW	Arrow that points to the upper-left corner of the screen.
SPTR_ICONERROR	Icon containing an exclamation point; used in a warning message box.
SPTR_ICONINFORMATION	Octagon-shaped icon containing the image of a human hand; used in a warning message box.
SPTR_ICONQUESTION	Icon containing a question mark; used in a query message box.
SPTR_ICONWARNING	Icon containing an asterisk; used in a warning message box.
SPTR_MOVE	Four-headed arrow; used when dragging an object or window around the screen.
SPTR_SIZE	Small box within a box; used when resizing a window by dragging.
SPTR_SIZENS	Two-headed arrow that points up and down (north and south); used when sizing a window.
SPTR_SIZENESW	Two-headed diagonal arrow that points to the upper-right (northeast) and lower-left (southwest) window borders; used when sizing a window.
SPTR_SIZENWSE	Two-headed diagonal arrow that points to the upper-left (northwest) and lower-right (southeast) window borders; used when sizing a window.
SPTR_SIZEWE	Two-headed arrow that points left and right (west to east); used when sizing a window.
SPTR_TEXT	Text-insertion and selection pointer, often called the <i>I-beam pointer</i> .
SPTR_WAIT	Hourglass; used to indicate that a time-consuming operation is in progress.

The operating system contains a second set of predefined mouse pointers that are used as icons in PM applications. An application can use one of these icons by supplying one of the following constants in WinQuerySysPointer. If a copy of the system pointer is made using WinQuerySysPointer, the pointer copy must be destroyed using WinDestroyPointer before termination of the application.

Icon	Description
SPTR_FILE	Represents a file (in the shape of a single sheet of paper).
SPTR_FOLDER	Represents a file folder.
SPTR_ILLEGAL	Circular icon containing a slash; represents an illegal operation.
SPTR_MULTFILE	Represents multiple files.
SPTR_PROGRAM	Represents an executable file.

Applications can use mouse-pointer resources to draw icons. WinDrawPointer draws a specified mouse pointer in a specified presentation space. Many of the predefined system mouse pointers are standard icons displayed in message boxes.

In addition to using the predefined pointer shapes, an application also can use pointers that have been defined in a resource file. Once the pointer or icon has been created (by Icon Editor or a similar application), the application includes it in the resource file, using the POINTER

statement, a resource identifier, and a file name for the Icon Editor data. After including the mouse-pointer resource, the application can use the pointer or icon by calling WinLoadPointer, specifying the resource identifier and module handle. Typically, the resource is in the executable file of the application, so the application simply can specify NULL for the module handle to indicate the current application resource file.

An application can create mouse pointers at run time by constructing a bit map for the pointer and calling WinCreatePointer. This function, if successful, returns the new pointer handle, which the application then can use to set or draw the pointer. The bit map must be twice as tall as it is wide, with the first half defining the AND mask and the second half defining the XOR mask. The application also must specify the hot spot when creating the mouse pointer.

System Bit Maps

In addition to using the mouse pointers and icons defined by the system, applications can use standard system bit maps by calling WinGetSysBitmap. This function returns a bit map handle that is passed to WinDrawBitmap or to one of the GPI bit-map functions. The system uses standard bit maps to draw portions of control windows, such as the system menu, minimize/maximize box, and scroll-bar arrows. The following standard system bit maps are available:

Bit Map	Description
SBMP_BTNCORNERS	Specifies the bit map for push button corners.
SBMP_CHECKBOXES	Specifies the bit map for the check-box or radio-button check mark.
SBMP_CHILDSYSMENU	Specifies the bit map for the smaller version of the system-menu bit map; used in child windows.
SBMP_CHILDSYSMENUDEF	Same as SBMP_CHILDSYSMENU but indicates that the system menu is selected.
SBMP_COMBODOWN	Specifies the bit map for the downward pointing arrow in a drop-down combination box.
SBMP_MAXBUTTON	Specifies the bit map for the maximize button.
SBMP_MENUATTACHED	Specifies the bit map for the symbol used to indicate that a menu item has an attached, hierarchical menu.
SBMP_MENUCHECK	Specifies the bit map for the menu check mark.
SBMP_MINBUTTON	Specifies the bit map for the minimize button.
SBMP_OLD_CHILDSYSMENU	Same as SBM_CHILDSYSMENU. (For compatibility with previous versions of the OS/2 operating system.)
SBMP_OLD_MAXBUTTON	Same as SBM_MAXBUTTON. (For compatibility with previous versions of the OS/2 operating system.)
SBMP_OLD_MINBUTTON	Same as SBM_MINBUTTON. (For compatibility with previous versions of the OS/2 operating system.)
SBMP_OLD_RESTOREBUTTON	Same as SBM_RESTOREBUTTON. (For compatibility with previous versions of the OS/2 operating system.)

SBMP_OLD_SBDNARROW	Same as SBM_SBDNARROW. (For compatibility with previous versions of the OS/2 operating system.)
SBMP_OLD_SBLFARROW	Same as SBM_SBLFARROW. (For compatibility with previous versions of the OS/2 operating system.)
SBMP_OLD_SBRGARROW	Same as SBM_SBRGARROW. (For compatibility with previous versions of the OS/2 operating system.)
SBMP_OLD_SBUPARROW	Same as SBM_SBUPARROW. (For compatibility with previous versions of the OS/2 operating system.)
SBMP_PROGRAM	Specifies the bit map for the symbol that File Manager uses to indicate that a file is an executable program.
SBMP_RESTOREBUTTON	Specifies the bit map for the restore button.
SBMP_RESTOREBUTTONDEP	Same as SBMP_RESTOREBUTTON but indicates that the restore button is pressed.
SBMP_SBDNARROW	Specifies the bit map for the scroll-bar down arrow.
SBMP_SBDNARROWDEP	Same as SBMP_SBDNARROW but indicates that the scroll-bar down arrow is pressed.
SBMP_SBDNARROWDIS	Same as SBMP_SBDNARROW but indicates that the scroll-bar down arrow is disabled.
SBMP_SBLFARROW	Specifies the bit map for the scroll-bar left arrow.
SBMP_SBLFARROWDEP	Same as SBMP_SBLFARROW but indicates that the scroll-bar left arrow is pressed.
SBMP_SBMFARROWDIS	Same as SBMP_SBLFARROW but indicates that the scroll-bar left arrow is disabled.
SBMP_SBRGARROW	Specifies the bit map for the scroll-bar right arrow.
SBMP_SBRGARROWDEP	Same as SBMP_SBRGARROW but indicates that the scroll-bar right arrow is pressed.
SBMP_SBRGARROWDIS	Same as SBMP_SBRGARROW but indicates that the scroll-bar right arrow is disabled.
SBMP_SBUPARROW	Specifies the bit map for the scroll-bar up arrow.
SBMP_SBUPARROWDEP	Same as SBMP_SBUPARROW but indicates that the scroll-bar up arrow is pressed.
SBMP_SBUPARROWDIS	Same as SBMP_SBUPARROW but indicates that the scroll-bar up arrow is disabled.
SBMP_SIZEBOX	Specifies the bit map for the symbol that indicates an area of a

	<p>window in which the user can click to resize the window.</p>
SBMP_SYSMENU	<p>Specifies the bit map for the system menu.</p>
SBMP_TREEMINUS	<p>Specifies the bit map for the symbol that File Manager uses to indicate an empty entry in the directory tree.</p>
SBMP_TREEPLUS	<p>Specifies the bit map for the symbol that File Manager uses to indicate that an entry in the directory tree contains more files.</p>

Using Mouse Pointers and Icons

This section explains how to perform the following tasks:

- Save the current mouse pointer
- Change the mouse pointer
- Restore the original mouse pointer

Changing the Mouse Pointer

Once you create or load a mouse pointer, you can change its shape by calling WinSetPointer. Following are three typical situations in which an application changes the shape of the mouse pointer:

- When an application receives a WM_MOUSEMOVE message, there is an opportunity to change the mouse pointer based on its location in the window. If you want the standard arrow pointer, pass this message on to WinDefWindowProc. If you want to change the mouse pointer on a standard dialog window, you need to capture the WM_CONTROLPOINTER message and return a pointing-device pointer handle.
- When an application is about to start a time-consuming process during which it will not accept user input, the application displays the *system-wait* mouse pointer (SPTR_WAIT). Upon finishing the process, the application resets the mouse pointer to its former shape.

The following code fragment shows how to save the current mouse pointer, set the hourglass pointer, and restore the original mouse pointer. Notice that the hourglass pointer also is saved in a global variable so that the application can return it when responding to a WM_MOUSEMOVE message during a time-consuming process.

```
HPOINTER hptrOld, hptrWait, hptrCurrent;

/* Get the current pointer. */
hptrOld = WinQueryPointer(HWND_DESKTOP);

/* Get the wait mouse pointer. */
hptrWait = WinQuerySysPointer(HWND_DESKTOP,
    SPTR_WAIT, FALSE);

/* Save the wait pointer to use in WM_MOUSEMOVE processing.*/
hptrCurrent = hptrWait;

/* Set the mouse pointer to the wait pointer. */
WinSetPointer(HWND_DESKTOP, hptrWait);

/*
 * Do a time-consuming operation, then restore the
 * original mouse pointer.
```

```
*/  
WinSetPointer(HWND_DESKTOP, hptrOld);
```

- When an application needs to indicate its current operational mode, it changes the pointer shape. For example, a paint program with a palette of drawing tools should change the pointer shape to indicate which drawing tool is in use currently.

Notebook Controls

A notebook control (WC_NOTEBOOK window class) is a visual component that organizes information on individual *pages* so that a user can find and display that information quickly and easily. This chapter explains how to use notebook controls in PM applications.

About Notebook Controls

This notebook control component simulates a real notebook but improves on it by overcoming a notebook's natural limitations. A user can select and display pages by using a pointing device or the keyboard.

The notebook can be customized to meet varying application requirements, while providing a user interface component that can be used easily to develop products that conform to the Common User Access (CUA) user interface guidelines. The application can specify different colors, sizes, and orientations for its notebooks and whether an old notebook or a new style notebook is desired, but the underlying function of the control remains the same.

Notebook Styles for an Old Notebook

This section describes the notebook style components associated with an old-style notebook:

- Page buttons
 - Major and minor tabs
 - Status line
 - Binding
 - Intersection of back pages
 - Tab shapes
-

Page Buttons

In the bottom-right corner of the notebook are the *page buttons*. These buttons let you bring one page of the notebook into view at a time. They are a standard component that is automatically provided with every old notebook. However, the application can change the default width and height of the page buttons by using the BKM_SETDIMENSIONS message. The page buttons always are located in the corner where the recessed edges of the notebook intersect.

Selecting the *forward page button* (the arrow pointing to the right) causes the next page to be displayed and selecting the *backward page button* (the arrow pointing to the left) causes the previous page to be displayed. Prior to inserting pages in the notebook, the page buttons are displayed with unavailable-state emphasis; therefore, selecting either page button would not bring a page into view.

Major Tabs

Major and *minor* tabs are used to organize related pages into sections. Minor tabs define subsections within major tab sections. The content of each section has a common theme, which is represented to the user by a tabbed divider that is similar to a tabbed page in a notebook, an address book, or a large dictionary.

The BKS_MAJORTABRIGHT style bit specifies that major tabs are to be placed on the right side of the notebook. This is the default major tab placement when the back pages intersect at the bottom-right corner of the notebook. The binding is located on the left, because it is

always located on the opposite side of the notebook from the major tabs.

The placement of the major tabs is limited to one of the two edges on which there are recessed pages. For example, if the application specifies the back pages intersection at the bottom-right corner (BKS_BACKPAGESBR, the default), the major tabs can be placed on either the bottom edge (BKS_MAJORTABBOTTOM) or the right edge (BKS_MAJORTABRIGHT) of the notebook. In this situation, if the application specifies that major tabs are to be placed on the left or top edges of the notebook, the notebook control places them on the right edge anyway-the default placement for back pages intersecting at the bottom-right corner.

When major tabs are defined at the creation of the notebook they are not displayed on screen. Major tab attributes only show at the time a page is inserted into the notebook. This is done by specifying the BKA_MAJOR attribute in the BKM_INSERTPAGE message.

Minor Tabs

Minor tabs are specified using the BKA_MINOR attribute. Minor tabs always are placed perpendicular to the major tabs, based on the intersection of the back pages and the major tab placement. Only one major or minor tab attribute can be specified for each notebook page. Minor tabs are displayed only if the associated major tab page is selected or if the notebook has no major tab pages.

The placement of the minor tabs depends entirely on the placement of the back pages and major tabs, respectively. The minor tabs always are located on the recessed page side that has no major tabs.

Status Line

To the left of the page buttons in the default old notebook style setting is the *status line*, which enables the application to provide information to the user about the page currently displayed. The notebook does not supply any default text for the status line.

The application is responsible for associating a text string with the status line of each page on which a text string is to be displayed. The status line can be used in addition to the major and minor tabs to provide the user with additional information on a particular page in the notebook. It is commonly used in large notebooks to assist the user in navigating, by indicating that this page is *page 1 of n*.

The status text is drawn left-justified by default, but it can be drawn centered or right-justified. The same status text justification applies to all pages in the notebook. This setting is specified by the BKS_STATUSTEXTLEFT style bit. The location of the back pages intersection and the major tabs has no effect on the specification of the status line position. This style bit can be set for the entire notebook.

Binding

The notebook control resembles a real notebook in its general appearance. The default binding is solid and is placed on the left side. This binding is used if the BKS_SOLIDBIND style bit is specified or if no style bit is specified.

Two styles are provided for the notebook binding: solid and spiral. The notebook is displayed with a solid binding by default, but the application can specify BKS_SPIRALBIND to display a spiral binding.

The placement of the binding depends entirely on the placement of the back pages and major tabs, respectively. The binding always is located on the opposite side of the notebook from the major tabs.

Intersection of Back Pages

The recessed edges that intersect near the page buttons are called the *back pages*. The default notebook's back pages intersect in the bottom-right corner, which means the recessed pages are on the bottom and right edges. This setting is specified by the BKS_BACKPAGESBR style bit. The back pages are important because their intersection determines where the major tabs can be placed, which in turn determines the placement of the binding and the minor tabs.

The following table describes the available notebook control styles for a old notebook:

Back Pages	Major Tabs	Minor Tabs	Binding
Bottom-right (default)	Bottom	Right	Top
Bottom-right (default)	Right (default)	Bottom	Left
Bottom-left	Bottom (default)	Left	Top
Bottom-left	Left	Bottom	Right
Top-right	Top (default)	Right	Bottom
Top-right	Right	Top	Left
Top-left	Top	Left	Bottom
Top-left	Left (default)	Top	Right

Tab Shapes and Contents

The default shape of the tabs used on notebook divider pages is square. This setting is specified by the `BKS_SQUARETABS` style bit. The shape of the tabs can be square, rounded, or polygonal. The tab text can be drawn left-justified, right-justified, or centered. Once set, these styles apply to the major and minor tabs for all pages in the notebook. The location of the back pages intersection and the major tabs has no effect on the specification of the tab-shape position. As with the page buttons, the application can change the default width and height of the major and minor tabs by using the `BKM_SETDIMENSIONS` message.

A notebook tab can contain text, a bitmap, or be owner-drawn. Text is associated with a tab page by using the `BKM_SETTABTEXT` message. Notebook tab text is centered by default or by specifying the `BKS_TABTEXTCENTER` style when creating the notebook window. A bitmap is placed on a tab by using the `BKM_SETTABBITMAP` message. Since the bitmap stretches to fill the rectangular area of the tab, no style bits are needed for bitmap positioning. If neither a `BKM_SETTABTEXT` nor a `BKM_SETTABBITMAP` message is used, the notebook owner is responsible for drawing the tab when a `WM_DRAWITEM` message is received for the tab.

Summary of Notebook Styles for Old Notebook

The old notebook control provides style bits so that your application can specify or change the default style settings. One style bit from each of the following groups can be specified. If you specify more than one style bit, you must use an OR operator (`|`) to combine them.

- Type of binding
 - BKS_SOLIDBIND Solid (default)
 - BKS_SPIRALBIND Spiral
- Intersection of back pages
 - BKS_BACKPAGESBR Bottom-right corner (default)
 - BKS_BACKPAGESBL Bottom-left corner
 - BKS_BACKPAGESTR Top-right corner
 - BKS_BACKPAGESTL Top-left corner
- Location of major tabs
 - BKS_MAJORTABRIGHT Right edge (default)
 - BKS_MAJORTABLEFT Left edge
 - BKS_MAJORTABTOP Top edge
 - BKS_MAJORTABBOTTOM Bottom edge
- Shape of tabs

<ul style="list-style-type: none"> <div> <div>BKS_SQUARETABS</div> <div>BKS_ROUNDEDTABS</div> <div>BKS_POLYGONTABS</div> </div> 	<div> <div>Square (default)</div> <div>Rounded</div> <div>Polygonal</div> </div>
<ul style="list-style-type: none"> Alignment of text associated with tabs <div> <div>BKS_TABTEXTCENTER</div> <div>BKS_TABTEXTLEFT</div> <div>BKS_TABTEXTRIGHT</div> </div> 	<div> <div>Centered (default)</div> <div>Left-justified</div> <div>Right-justified</div> </div>
<ul style="list-style-type: none"> Alignment of status-line text <div> <div>BKS_STATUSTEXTLEFT</div> <div>BKS_STATUSTEXTRIGHT</div> <div>BKS_STATUSTEXTCENTER</div> </div> 	<div> <div>Left-justified (default)</div> <div>Right-justified</div> <div>Centered</div> </div>
<ul style="list-style-type: none"> Styles not applicable to an Old Notebook <div> <div>BKS_TABBEDDIALOG</div> <div>BKS_BUTTONAREA</div> </div> 	<div> <div>Indicates a new-style notebook is desired.</div> <div>Creates common button area for a new notebook. (This style uses the same area as BKS_POLYGONTABS, so it may generate unexpected results if used.)</div> </div>

Notebook Styles for a New Notebook

This section describes the notebook style components associated with a new-style notebook:

- Dog-eared tab
- Major tabs
- Scroll tabs
- Status line
- Minor tabs and the Page List
- Common button area

A new-style notebook will reformat itself if a new font is dropped on the notebook window.

Dog-Eared Tab

The page buttons in the old notebook control have been replaced by a dog-eared tab in the upper-right hand corner of the control. Selecting the area behind the dog-eared tab causes the next page to be displayed and selecting the dog-eared tab itself causes the previous page to be displayed. Plus and minus symbols emphasize the directions the user can travel in the notebook. A + symbol in the area behind the dog-eared tab indicates the user can page forward, and a - symbol in the dog-eared tab indicates the user can page backward. The symbol disappears when there are no more pages to turn in that direction.

Major Tabs

Major tabs are used to organize related pages in sections. Major tabs in a new notebook may appear along the bottom of the notebook by specifying the BKS_MAJORTABBOTTOM style, or they may appear along the top of the notebook by default or by explicitly specifying the BKS_MAJORTABTOP style. The application can control whether the tabs are ordered from left to right or right to left by using the appropriate style. Minor tabs are no longer displayed as *tabs* on the new notebook. Minor tabs are shown as part of the page list. The following table describes the available notebook control styles:

Style	Left to Right (default)	Right to Left
BKS_MAJORTABTOP (default)	BKS_BACKPAGESTR	BKS_BACKPAGESTL
BKS_MAJORTABBOTTOM	BKS_BACKPAGESBR	BKS_BACKPAGESBL

Status line

To the left of the dog-eared tab is the status line, which enables the application to provide information to the user about the page being displayed. The notebook does not supply any default text for this line, thus the application is responsible for associating a text string with the status line on each page of the notebook.

The status line is always next to the dog-eared tab, thus the BKS_STATUSTEXTLEFT, BKS_STATUSTEXTRIGHT, and BKS_STATUSTEXTMIDDLE styles are ignored.

Left and Right Scroll Tabs

Left and right scroll tabs appear as necessary on either side of the major tabs. Click on the appropriate scroll tab until the tab you want scrolls into view.

Page List

The page list is a popup menu that lists all the pages in the notebook. To display the page list, position the pointer on a notebook tab or border and click mouse button 2. If a major tab in the page list has more than one page associated with it, its pages are placed in a submenu.

The text displayed in the page list for a given page is determined by the minor tab, major tab, and status text associated with the page. If a minor tab is associated with a page, its text is used. If only a major tab is associated with a page, the major tab text is used. If no tab is associated with the page, the status line text is used for the page list. If no status text is provided for the page, the page list entry is left blank.

Common Button Area

When the BKS_BUTTONAREA style is set, a common button area is created to contain a set of push buttons which is the same on each notebook page. These buttons are children of the notebook window. Buttons common to all pages in the notebook can be added, and specific buttons for a given notebook page may also be added as needed. The button area is placed opposite the major tabs of the notebook.

Common buttons behave exactly like ordinary push buttons except that they are positioned and sized automatically by the notebook control. Notification messages from the buttons are sent to the current notebook page. Buttons common to all notebook pages are handled by the BKM_SETNOTEBOOKBUTTONS message. Buttons specific to a notebook page can be created as children of that notebook page using the BS_NOTEBOOKBUTTON style.

If you create BS_NOTEBOOKBUTTON style buttons for a notebook page, they will override the common buttons created by BKM_SETNOTEBOOKBUTTONS. For instance, if you want to have buttons **Help** and **Default** available on all pages but one, use the BKM_SETNOTEBOOKBUTTONS message to create them, but create BS_NOTEBOOKBUTTON style buttons on the page requiring special buttons, and they will be the ones shown when that particular notebook page is displayed.

The states of the buttons created using BKM_SETNOTEBOOKBUTTONS are not changed when the notebook page turns.

Summary of Notebook Styles for New Notebook

The new notebook control provides style bits so that your application can specify or change the default style settings. One style bit from each of the following groups can be specified. If you specify more than one style bit, you must use an OR operator (|) to combine them.

- New Notebook
 - BKS_TABBEDDIALOG
 - Indicates a new notebook control is desired. (The default is an old notebook.)
- Location of major tabs
 - BKS_MAJORTABTOP
 - Top edge (default)
 - BKS_MAJORTABBOTTOM
 - Bottom edge
- Ordering of major tabs
 - BKS_BACKPAGESTR
 - Left-to-right when BKS_MAJORTABTOP (default)
 - BKS_BACKPAGESTL
 - Right to left when BKS_MAJORTABTOP
 - BKS_BACKPAGESBR
 - Left-to-right when BKS_MAJORTABBOTTOM (default)
 - BKS_BACKPAGESBL
 - Right-to-left when BKS_MAJORTABBOTTOM
- Button Area
 - BKS_BUTTONAREA
 - Creates space above or below the notebook page for a set of push buttons.
- Styles ignored by New Notebook
 - BKS_SOLIDBIND
 - Solid binding
 - BKS_SPIRALBIND
 - Spiral binding
 - BKS_TABTEXTCENTER
 - Centered tab text
 - BKS_TABTEXTLEFT
 - Left-justified tab text
 - BKS_TABTEXTRIGHT
 - Right-justified tab text
 - BKS_STATUSTEXTLEFT
 - Left-justified status text
 - BKS_STATUSTEXTRIGHT
 - Right-justified status text
 - BKS_STATUSTEXTCENTER
 - Centered status text
 - BKS_MAJORTABRIGHT
 - Major tabs on right edge
 - BKS_MAJORTABLEFT
 - Major tabs on left edge
 - BKS_SQUARETABS
 - Square tabs
 - BKS_ROUNDEDTABS
 - Rounded tabs
 - BKS_POLYGONTABS
 - Polygonal tabs (Do not specify with BKS_TABBEDDIALOG because this style overlaps BKS_BUTTONAREA and you might get unexpected results.)

Using Notebook Controls

The following sections describe how to create pages, insert pages into, create and associate windows for, and delete pages from a notebook.

Notebook Creation

You create a notebook by using the WC_NOTEBOOK window class name in the *ClassName* parameter of WinCreateWindow. The following sample code shows the creation of the notebook. The style set in the *ulNotebookStyles* variable (the BKS_* values) specifies that the notebook is to be created with a solid binding and the back pages intersecting at the bottom-right corner, major tabs placed on the right edge, shape tab square, tab text centered, and status-line text left-justified. These are the default settings and are given here only to show how notebook styles are set.

```
HWND    hwndNotebook;           /* Notebook window handle          */
ULONG    ulNotebookStyles;       /* Notebook window styles          */
ULONG    x, y, cx, cy;          /* Coordinates                      */

/*****
/* Set the BKS_style flags to customize the notebook.
*****/
ulNotebookStyles =
    BKS_SOLIDBIND |             /* Use solid binding                */
    BKS_BACKPAGESBR |          /* Set back pages to intersect at the */
                                /* bottom-right corner              */
    BKS_MAJORTABRIGHT |        /* Position major tabs on right side */
    BKS_SQUARETABS |           /* Make tabs square                 */
    BKS_TABTEXTCENTER |        /* Center tab text                  */
    BKS_STATUSTEXTLEFT;        /* Left-justified status-line text  */

/*****
/* Create the notebook control window.
*****/
hwndNotebook =
    WinCreateWindow(
        hwndParent,             /* Parent window handle            */
        WC_NOTEBOOK,           /* Notebook window class           */
        NULL,                   /* No window text                  */
        ulNotebookStyles,       /* Notebook window styles          */
        x, y, cx, cy,           /* Origin and size                 */
        hwndOwner,              /* Owner window handle             */
        HWND_TOP,               /* Sibling window handle          */
        ID_BOOK,                /* Notebook window ID              */
        NULL,                   /* No control data                 */
        NULL);                  /* No presentation parameters     */

/*****
/* Make the notebook control visible.
*****/
WinShowWindow(
    hwndNotebook,              /* Notebook window handle          */
    TRUE);                     /* Make the window visible        */
```

Changing Notebook Styles

The following figure shows some sample code fragments for setting the notebook style to spiral binding, back pages intersecting at the bottom-left corner, major tabs placed on the bottom edge, tab shape rounded, tab text left-justified, and status-line text centered.

```
/*****
/* Query for the existing notebook window style settings.
*****/
ulNotebookStyles =
    WinQueryWindowULONG(hwndNotebook, /* Notebook window handle          */
        QWL_STYLE);                  /* Set notebook style              */

/*****
/* Reset notebook window style flags, leaving window flags unchanged.
*****/
```

```

/*****
ulNotebookStyles &= 0xFFFF0000;

/*****
/* Setup the new notebook window style flags. */
/*****
ulNotebookStyles |=
    BKS_SPIRALBIND      |      /* Use spiral binding          */
    BKS_BACKPAGESBL     |      /* Set back pages to intersect at the */
                                /* bottom-left corner          */
    BKS_MAJORTABBOTTOM  |      /* Position major tabs on bottom edge */
    BKS_ROUNDEDTABS     |      /* Make tabs rounded              */
    BKS_TABTEXTLEFT     |      /* Left-justified tab text         */
    BKS_STATUSTEXTCENTER;      /* Center status-line text         */

/*****
/* Set the new notebook style. */
/*****
WinSetWindowULong(
    hwndNotebook,      /* Notebook window handle */
    QWL_STYLE,         /* Window style            */
    ulNotebookStyles); /* Set notebook style      */

/*****
/* Invalidate to force a repaint. */
/*****
WinInvalidateRect(
    hwndNotebook,      /* Notebook window handle */
    NULL,              /* Invalidate entire window, */
    TRUE);             /* including children       */

```

Inserting Notebook Pages

After a notebook is created, pages can be inserted into the notebook by using the BKM_INSERTPAGE message. BKM_INSERTPAGE provides several attributes that can affect the inserted pages. When inserting pages into either a new notebook or an existing one, consider carefully how the user expects those pages to be organized.

Major and Minor Tabs

The two attributes that have the most impact on how notebook pages are organized are BKA_MAJOR and BKA_MINOR, which specify major and minor tabs, respectively. Major tab pages define the beginning of major sections in the notebook, while minor tab pages define the beginning of subsections within a major section. Major sections should begin with a page that has a BKA_MAJOR attribute. Within major sections, information can be organized into minor sections, each of which should begin with a page that has a BKA_MINOR attribute.

For an existing notebook, the underlying hierarchy, if one exists, must be observed when inserting new pages, to provide efficient organization and navigation of the information in the notebook. For example, if the notebook has minor sections but no major sections, you could confuse the user if you inserted a page with a major tab attribute between related minor sections or at the end of the notebook.

If you insert pages without specifying tab attributes, those pages become part of the section in which they are inserted. For example, if page 7 of your notebook has a minor tab and you insert a new page 8 without specifying a tab attribute, page 8 becomes part of the section that begins with the minor tab on page 7.

Because tab pages are not mandatory, the application can create a notebook that contains no major or minor tab pages. That style would be similar to that of a composition notebook.

Another group of attributes that can affect the organization of pages being inserted into a notebook consists of BKA_FIRST, BKA_LAST, BKA_NEXT, and BKA_PREV. These attributes cause pages to be inserted at the end, at the beginning, after a specified page, and before a specified page of a notebook, respectively.

Status Line

Each page has an optional status line that can be used to display information for the user. To include this status line, the application must specify the `BKA_STATUSTEXTON` attribute when inserting the page. If the application inserts the page without specifying this attribute, the status line is not available for that page.

To display text on the status line of the specified page, the application must use the `BKM_SETSTATUSLINETEXT` message to associate a text string with the page. A separate message must be sent for each page that is to display status-line text. If the application does not send a `BKM_SETSTATUSLINETEXT` message for a page, no text is displayed in the status line of that page. The application can send this message to the notebook at any time to change the status-line text. The status line can be cleared by setting the text to `NULL`.

The following sample code shows how to insert a page into a notebook, where the inserted page has a major tab attribute, the status line is available, and the page is inserted after the last page in the notebook. This sample code also shows how to associate a text string with the status line of the inserted page.

```
HWND  hwndNotebook;          /* Notebook window handle          */
ULONG ulPageId;              /* Page identifier                  */

/*****
/* Insert a new page into a notebook.
*****/
ulPageId = (ULONG) WinSendMessage(
    hwndNotebook,             /* Notebook window handle          */
    BKM_INSERTPAGE,           /* Message for inserting a page    */
    (MPARAM) NULL,            /* NULL for page ID                */

    MPFROM2SHORT(
        BKA_MAJOR |           /* Insert page with a major tab    */
        BKA_STATUSTEXTON),     /* Make status-line text visible   */
    BKA_LAST));               /* Insert this page at end of notebook

/*****
/* Set the status-line text.
*****/
WinSendMessage(
    hwndNotebook,             /* Notebook window handle          */
    BKM_SETSTATUSLINETEXT,    /* Message for setting status-line */
    (MPARAM) ulPageId,        /* ID of page to receive status-line */
    MPFROMP("Page 1 of 2"));  /* Text string to put on status line
```

Setting and Querying Page Information

The information for a page in the notebook can be set and queried with `BKM_SETPAGEINFO` and `BKM_QUERYPAGEINFO` respectively. By using these messages, all the information associated with a page can be accessed at once. In addition, `BKM_SETPAGEINFO` can be used to delay the loading of a page until it is turned to, by setting the *bLoadDlg* field to `FALSE`. By doing this for all pages in a notebook, the notebook is created much more quickly.

Associating Application Page Windows with Notebook Pages

After a page is inserted into a notebook, you must facilitate the display of information for this page when it is brought to the top of the book. The notebook provides a top page area in which the application can display windows or dialogs for the topmost page. For each inserted page, the application must associate the handle of a window or dialog that is to be invalidated when the page is brought to the top of the book. The application can associate the same handle with different pages, if desired.

The application must send a BKM_SETPAGEWINDOWHWND message to the notebook in order to associate the application page window or dialog handle with the notebook page being inserted. Once done, the notebook invalidates this window or dialog whenever the notebook page is brought to the top of the book. If no application page window handle is specified for an inserted page, no invalidation can be done by the notebook for that page. However, the application receives a BKN_PAGESELECTED notification code when a new page is brought to the top of the notebook, at which time the application can invalidate the page.

The notebook also sends a BKN_PAGESELECTEDPENDING notification code to the application before the new page is selected. The application can use this message to prevent the page from being turned to. This is useful when the application wants to validate a page's contents.

The following sections describe how to associate either a window handle or a dialog handle with an inserted page.

Associating a Window with a Notebook Page

A calendar example is used to show how a page can be implemented as a window. A calendar is divided into four years (major tabs). Within each year are months (minor tabs) grouped into quarters. The top page has a window associated with it. The window paint processing displays the days for the currently selected month and year.

The sample code in the following figure shows how the window procedure for the calendar is registered with the application. Also, it shows how the window is created and associated with the notebook page. The example ends by showing the window procedure for the associated window.

```

/*****
/* Registration of window procedure for calendar.
*****/
WinRegisterClass(hab, /* Register a page window class */
    "Calendar Page", /* Class name */
    PageWndProc, /* Window procedure */
    CS_SIZEREDRAW, /* Class style */
    0); /* No extra bytes reserved */

/*****
/* Create the window.
*****/
hwndPage = WinCreateWindow(hwndNotebook, /* Parent */
    "Calendar Page", /* Class */
    NULL, /* Title text */
    0L, /* Style */
    0, 0, 0, 0, /* Origin and size */
    hwndNotebook, /* Owner */
    HWND_TOP, /* Z-order */
    ID_WIN_CALENDAR_PAGE, /* ID */
    NULL, /* Control data */
    NULL); /* Presparams */

/*****
/* Associate window with the inserted notebook page.
*****/
WinSendMsg(hwndBook,
    BKM_SETPAGEWINDOWHWND,
    MPFROMLONG(ulPageId),
    MPFROMHWND(hwndPage));

/*****
/* Window procedure.
*****/
MRESULT EXPENTRY PageWndProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
    HPS hps;

    switch (msg)
    {
        /*****
        /* WM_CREATE is sent when the window is created.
        *****/
        case WM_CREATE:

```



```

                                /* address */
                                /* Dialog data structure */
                                /* address */
                                /* Application data */
                                /* Free memory */

pDlgt,
NULL);

DosFreeMem(pDlgt);

/*****
/* Associate dialog with the inserted notebook page.
*****/
WinSendMsg(hwndBook,
            BKM_SETPAGEWINDOWHWND,
            MPFROMLONG(ulPageId),
            MPFROMHWND(hwndPage));

/*****
/* Dialog procedure.
*****/
/*****
MRESULT EXPENTRY fnwpPrint(HWND hwndDlg,ULONG msg,MPARAM mp1,MPARAM mp2)
{
    switch (msg)
    {
        case WM_INITDLG:

            /*****
            /* Place dialog initialization code here.
            *****/
            break;

        case WM_COMMAND:
            return ((MRESULT) FALSE);
            break;

        default:
            return WinDefDlgProc (hwndDlg,msg,mp1,mp2);
    }
    return WinDefDlgProc (hwndDlg,msg,mp1,mp2);
}

```

Deleting Notebook Pages

The BKM_DELETEPAGE message is used to delete one or more pages from the notebook. The application can delete one page (BKA_SINGLE attribute), all pages within a major or minor tab section (BKA_TAB attribute), or all of the pages in the notebook (BKA_ALL attribute). The default, if no attributes are specified, is to delete no pages. The following sample code shows how the BKM_QUERYPAGEID message is used to get the ID of the top page and how the BKM_DELETEPAGE message is then used to delete that page:

```

/*****
/* Set the range of pages to be deleted.
*****/

/* Set attribute to delete a single page. */
usDeleteFlag = BKA_SINGLE

/*****
/* Get the ID of the notebook's top page.
*****/
ulPageId = (ULONG) WinSendMsg(
    hwndNotebook,          /* Notebook window handle */
    BKM_QUERYPAGEID,       /* Message to query a page ID */
    NULL,                  /* NULL for page ID */
    (MPARAM)BKA_TOP);      /* Get ID of top page */

/*****
/* Delete the notebook's top page.
*****/
WinSendMsg(
    hwndNotebook,          /* Notebook window handle */
    BKM_DELETEPAGE,        /* Message to delete the page */

```

```

MPFROMLONG(ulPageId),      /* ID of page to be deleted      */
(MPARAM)usDeleteFlag);    /* Range of pages to be deleted */

```

Notebook Colors

The application can change the color of any part of the notebook. The colors of some parts can be changed by specifying presentation parameter attributes in WinSetPresParam. Other colors can be changed by specifying notebook attributes in the BKM_SETNOTEBOOKCOLORS message. The following sections define which parts of the notebook can have their colors changed by each of these two methods.

Changing Colors Using WinSetPresParam

WinSetPresParam is used to change the color of the notebook outline, window background, selection cursor, and status-line text. The following list shows the mapping between the various notebook parts and their associated presentation parameter attributes.

Notebook outline

PP_BORDERCOLOR or PP_BORDERCOLORINDEX. This color is set initially to SYSCLR_WINDOWFRAME.

Notebook window background

PP_BACKGROUNDCOLOR or PP_BACKGROUNDCOLORINDEX. This color is set initially to SYSCLR_FIELDBACKGROUND.

Selection cursor

PP_HILITEBACKGROUNDCOLOR or PP_HILITEBACKGROUNDCOLORINDEX. This color is set initially to SYSCLR_HILITEBACKGROUND.

Status-line text

PP_FOREGROUNDCOLOR or PP_FOREGROUNDCOLORINDEX. This color is initially set to SYSCLR_WINDOWTEXT.

If a presentation parameter attribute is set, all parts of the notebook that are mapped to this color are changed. The following sample code shows how to change the color of the notebook outline:

```

/* Set number of bytes to be passed in usColorIdx */
/* for color index table value                      */
usColorLen = 4;
/* Set color index table value to be assigned */
ulColorIdx = 3;

/*****
/* Set the notebook outline color.                      */
*****/
WinSetPresParam(
    hwndNotebook,      /* Notebook window handle      */
    PP_BORDERCOLOR,    /* Border color attribute       */
    usColorLen,        /* Number of bytes in color index */
    /* table value */
    &ulColorIdx);      /* Color index table value     */

```

Changing Colors Using BKM_SETNOTEBOOKCOLORS

The BKM_SETNOTEBOOKCOLORS message is used to change the color of the major tab background and text, the minor tab background and text, and the notebook page background. The following list shows the mapping between the various notebook parts and their associated

notebook attributes.

Major tab background

BKA_BACKGROUNDMAJORCOLOR or BKA_BACKGROUNDMAJORCOLORINDEX. This color is set initially to SYSCLR_PAGEBACKGROUND. The currently selected major tab has the same background color as the notebook page background.

Major tab text

BKA_FOREGROUNDMAJORCOLOR or BKA_FOREGROUNDMAJORCOLORINDEX. This color is set initially to SYSCLR_WINDOWTEXT.

Minor tab background

BKA_BACKGROUNDMINORCOLOR or BKA_BACKGROUNDMINORCOLORINDEX. This color is set initially to SYSCLR_PAGEBACKGROUND. The currently selected minor tab has the same background color as the notebook page background.

Minor tab text

BKA_BACKGROUNDMINORCOLOR or BKA_BACKGROUNDMINORCOLORINDEX. This color is set initially to SYSCLR_WINDOWTEXT.

Notebook page background

BKA_BACKGROUNDPAGECOLOR or BKA_BACKGROUNDPAGECOLORINDEX. This color is set initially to SYSCLR_PAGEBACKGROUND.

If a notebook attribute is set, all parts of the notebook that are mapped to this color are changed. The following sample code shows how to change the color of the major tab background:

```
/* Color index value      */
ulColorIdx    = SYSCLR_WINDOW;
/* Major tab background */
ulColorRegion = BKA_BACKGROUNDMAJORCOLORINDEX;

WinSendMsg(hwndBook,
            BKM_SETNOTEBOOKCOLORS,
            MPFROMLONG(ulColorIdx),
            MPFROMLONG(ulColorRegion));
```

Graphical User Interface Support for Notebook Controls

The following section describes the support for graphical user interfaces (GUIs) provided by the notebook control. Except where noted, this support conforms to the guidelines in the *SAA CUA Advanced Interface Design Reference*.

The GUI support provided by the notebook control consists of the notebook navigation techniques.

Notebook Navigation Techniques

The notebook control supports the use of a pointing device and the keyboard for displaying notebook pages and tabs and for moving the selection cursor from the notebook tabs to the application window and the other way around.

Note: If more than one notebook window is open, displaying a page or tab in one notebook window has no effect on the pages or tabs displayed in any other notebook window.

Pointing Device Support

A user can use a pointing device to display notebook pages or tabs by selecting the notebook components described in the following list. The CUA guidelines define mouse button 1 (the select button) to be used for selecting these components. This definition also applies to the same button on any other pointing device a user might have.

- Selecting tabs using a pointing device

A tab can be selected to bring a page that has a major or minor tab attribute to the top of the notebook. The *selection cursor*, a dotted outline, is drawn inside the tab's border to indicate the selected tab. In addition, the selected tab is given the same background color as the notebook page area. The color of the other tabs is specified in the BKM_SETNOTEBOOKCOLORS message. This helps the user distinguish the selected tab from the other tabs if different colors are used.

Because all tabs are mutually exclusive, only one of them can be selected at a time. Therefore, the only type of selection supported by the notebook control is *single selection*. This selection type conforms to the guidelines in the *SAA CUA Advanced Interface Design Reference*.

If the user moves the pointing device to a place in the notebook page window that can accept a cursor, such as an entry field, check box, or radio button, and presses the select button, the selection cursor is removed from the tab it is on and is displayed in the notebook page window. The selection cursor never can be displayed both on a tab and in the notebook page window at the same time.
- Selecting page buttons using a pointing device

A forward or backward *page button* can be selected to display the next or previous page, respectively, one at a time. The arrow pointing to the right is the forward page button, and the arrow pointing to the left is the backward page button. When the selection of a page button brings a page that has a major or minor tab to the top of the notebook, the selection cursor is drawn inside that tab's border.
- Selecting tab scroll buttons using a pointing device

A user can decrease the size of a notebook window so that some of the available notebook tabs cannot be displayed. When this happens, the notebook control automatically draws *tab scroll buttons* at the corners of the notebook side or sides to notify the user that more tabs are available.

Tab scroll buttons have another purpose: to give the user the means to scroll into view, one at a time, the tabs that are not displayed. The user does this by selecting a forward or backward tab scroll button, which causes the next tab to scroll into view, but does not change the location of the selection cursor. Once the tab is in view, the user can display that tab's page by selecting the tab.

A maximum of four tab scroll buttons can be displayed: two for the major tab side and two for the minor tab side.

When the first tab in the notebook is displayed, the backward tab scroll button is deactivated. Unavailable-state emphasis is applied to it to show that no more tabs can be scrolled into view by using the backward tab scroll button. Unavailable-state emphasis is applied to the forward tab scroll button if the last tab in the notebook is displayed.

Keyboard Support

The users can utilize the keyboard to display and manipulate notebook pages and components.

Focus on Application Dialog or Window

If the application dialog page or window has the focus, the notebook handles the following keyboard interactions:

Keyboard Input	Description
Alt+PgDn or PgDn	Brings the next page to the top of the notebook. If the application uses the PgDn key, then it must be used in combination with the Alt key.
Alt+PgUp or PgUp	Brings the previous page to the top of the notebook. If the application uses the

Alt+Up Arrow	PgUp key, then it must be used in combination with the Alt key.
Tab	Switch the focus to the notebook window.
Shift+Tab	Move the cursor to the next control within the top page window or dialog. If the cursor is currently on the last control within the top page window or dialog when the Tab key is pressed, the cursor is moved to the notebook major tab, if it exists; else to the minor tab, if it exists; else to the right page button.
	Move the cursor to the previous control within the top page window or dialog. If the cursor is currently on the first control within the top page window or dialog when the Shift+Tab key is pressed, the cursor is moved to the previous control. If the previous control is the notebook, the cursor is moved to the right page button.

Focus on the Notebook Control

If the notebook control has the focus, it handles the following keyboard interactions:

Keyboard Input	Description
Alt+Down Arrow	Switch the focus to the application's primary window.
Alt+PgDn or PgDn	Brings the next page to the top of the notebook.
Alt+PgUp or PgUp	Brings the previous page to the top of the notebook.
Left or Up Arrow	<p>If the cursor is currently on a major tab, it is moved to the previous major tab. If the previous major tab is not visible, the tabs are scrolled to bring the previous major tab into view. If the first major tab is reached, scrolling ends.</p> <p>If the cursor is currently on a minor tab, it is moved to the previous minor tab. If the previous minor tab is not visible, the tabs are scrolled to bring the previous minor tab into view. If the first minor tab is reached, scrolling ends.</p> <p>If the cursor is currently on the right page button, the cursor moves to the left page button. If the cursor is currently on the left page button, no action is taken.</p>
Right or Down Arrow	<p>If the cursor is currently on a major tab, it is moved to the next major tab. If the next major tab is not visible, the tabs are scrolled to bring the next major tab into view. If the last major tab is reached, scrolling ends.</p> <p>If the cursor is currently on a minor tab, it is moved to the next minor tab. If the next minor tab is not visible, the tabs are scrolled to bring the next minor tab into view. If the last minor tab is reached, scrolling ends.</p> <p>If the cursor is currently on the right page button, no action is taken. If the cursor is currently on the left page button, the cursor moves to the right page button.</p>
Tab	The cursor moves from the major tab, then to the minor tab, then to the right page button, and then to the last tab stop in the application dialog or window.
Shift+Tab	The cursor moves from the page button, to the minor tab, to the major tab, and then to the first tab stop in the application dialog or window.
Home	Brings the first page of the notebook to the top and sets the cursor on the associated tab.
End	Brings the last page of the notebook to the top and sets the cursor on the associated tab.
Enter or Spacebar	If the cursor is on a major or minor tab, the associated page is brought to the top of the notebook, and the selected tab is given the same background color as the notebook page area. The other tabs have their color specified in the BKM_SETNOTEBOOKCOLORS message. This helps the user distinguish the selected tab from the other tabs if different colors are used.

If the cursor is currently on the right page button, the next page is brought to the top of the notebook. If the cursor is currently on the left page button, the previous page is brought to the top of the notebook.

Mnemonics

Mnemonics are underlined characters in the text of a tab that cause the tab's page to be selected. Coding a tilde (~) before a text character in the BKM_SETTABTEXT message causes that character to be underlined and activates it as a mnemonic-selection character.

A user performs mnemonic selection by pressing a character key that corresponds to an underlined character. When this happens, the tab that contains the underlined character is selected, and that tab's page is brought to the top of the notebook.

Note: Mnemonic selection is not case sensitive, so the user can type the underscored letter in either uppercase or lowercase.

Enhancing Notebook Control Performance and Effectiveness

This section provides the following information to enable you to fine-tune a notebook control:

- Dynamic resizing and scrolling
- Tab painting and positioning

Dynamic Resizing and Scrolling

The notebook control supports *dynamic resizing* by recalculating the size of the notebook's parts when either the user or the application changes the size of any of those parts. A BKN_NEWPAGESIZE notification code is sent from the notebook to the application whenever the notebook's size changes.

The notebook handles the sizing and positioning of each application page window if the BKA_AUTOPAGESIZE attribute is specified for the inserted notebook page. Otherwise, the application must handle this when it receives the BKN_NEWPAGESIZE notification code from the notebook.

If the size of the notebook window is decreased so that the page window is not large enough to display all the information the page contains, the information in the page window is clipped. If scroll bars are desired to enable the clipped information to be scrolled into view, they must be provided by the application. Tab scroll buttons are automatically displayed if the size of the notebook is decreased so that all the major or minor tabs cannot be displayed. For example, a notebook has major tabs on the right side, but the height of the notebook does not allow all the tabs to be displayed. In this case, tab scroll buttons are displayed on the upper- and lower-right corners of the notebook.

Tab Painting and Positioning

The tab pages provide a method for organizing the information in a notebook so that the user easily can see and navigate to that information. When a page is inserted with a major or minor tab attribute, the notebook displays a tab for that page, based on the orientation of the notebook. The contents of the tab can be painted either by the notebook control or the application.

If the notebook control is to paint the tabs, the application must associate a text string or bit map with the page whose tab is to be drawn. This is done by sending the BKM_SETTABTEXT or BKM_SETTABBITMAP message to the notebook control for the specified page. If neither of these messages is sent for an inserted page with a major or minor tab attribute, the application must draw the contents of the tab, through *ownerdraw*. The application receives a WM_DRAWITEM message whenever a tab page that has no text or bit map associated with it is to be drawn. The application can either draw the tab contents or return FALSE, in which case the notebook control fills the tab with the tab background color.

Positioning Tabs in Relation to the Top Tab:

There are seven page edges that define the back pages. The page attribute (BKA_MAJOR or BKA_MINOR) and the topmost page determine how the tabs are positioned. In most cases, the tabs must be drawn when their position changes. For example, this can happen when a page with a tab attribute is brought to the top of the notebook.

The new top major or minor tab will appear attached to the top page. The other tabs will appear as described in the following list. This information is provided to help you understand the relationship between the top tab and the other tabs so that you can organize the information you put into a notebook appropriately. The application has no control over tab positioning.

- When the top page is a major tab page:
 - Any major tabs prior to the top major tab are aligned on the last page of the notebook.
 - Any major tabs after the top major tab are incrementally cascaded from the topmost edge to the last page.
 - If the top major tab has minor tabs, no major tab is drawn on the page edge that immediately follows the top tab page. Instead, any major tabs that follow the top tab are incrementally cascaded, beginning on the second page, edge-down from the top tab. This is done to account for the minor tabs that are positioned between the top major tab and the major tab that follows it on the perpendicular notebook edge.

The minor tabs are all positioned on the third page edge from the top, thereby giving the appearance of being between the top major tab and the next major tab.
- When the top page is a minor tab page:
 - Any minor tabs prior to the top minor tab are positioned on the third page edge from the top of the notebook.
 - Any minor tabs after the top minor tab are incrementally cascaded up to the third page edge from the top.

Painting and Drawing

This chapter describes presentation spaces, device contexts, and window regions, explaining how a PM application uses them for painting and drawing in windows.

About Painting and Drawing

An application typically maintains an internal representation of the data that it is manipulating. The information displayed in a screen, window, or printed copy is a visual representation of some portion of that data. This chapter introduces the concepts and strategies necessary to make your PM application function smoothly and cooperatively in the OS/2 display environment.

Presentation Spaces and Device Contexts

A *presentation space* is a data structure, maintained by the operating system, that describes the drawing environment for an application. An application can create and hold several presentation spaces, each describing a different drawing environment. All drawing in a PM application must be directed to a presentation space.

Normally each presentation space is associated with a *device context* that describes the physical device where graphics commands are displayed. The device context translates graphics commands made to the presentation space into commands that enable the physical device to display information. Typical device contexts are the screen, printers and plotters, and off-screen memory bit maps. By creating presentation spaces and associating them with particular device contexts, an application can control where its graphics output appears. Typically, a presentation space and device context isolate the application from the physical details of displaying graphics, so the same graphics commands can be used for many types of displays. This virtualization of output can reduce the amount of display code an application must include to support multiple output devices.

This chapter describes how an application sets up its presentation spaces and device contexts before drawing, and how to use window-drawing functions. Refer to the *Graphics Programming Interface Programming Guide* for the graphics functions available to PM applications.

Window Regions

A window and its associated presentation space have three regions that control where drawing takes place in the window. These regions ensure that the application does not draw outside the boundaries of the window or intrude into the space of an overlapping window.

Region	Description
Update Region	This region represents the area of the window that needs to be redrawn. This region changes when overlapping windows change their z-order or when an application explicitly adds an area to the update region to force a window to be painted.
Clip Region	This region and the visible region determine where drawing takes place. Applications can change the clip region to limit drawing to a particular portion of a window. Typically, a presentation space is created with a clip region equal to NULL, which makes this region equivalent to the update region.
Visible Region	This region and the clip region determine where drawing takes place. The system changes the visible region to represent the portion of a window that is visible. Typically, the visible region is used to mask out overlapping windows. When an application calls the WinBeginPaint function in response to a WM_PAINT message, the system sets the visible region to the intersection of the visible region and the update region to produce a new visible region. Applications cannot change the visible region directly.

Whenever drawing occurs in a window's presentation space, the output is clipped to the intersection of the visible region and clip region.

The clip region includes the overlapped part of the back window, but the visible region excludes that portion of the back window. The system maintains the visible region to protect other windows on the screen; the application maintains the clip region to specify the portion of the window in which it draws. Together, these two regions provide safe and controllable clipping.

To further control drawing, both the system and the application manipulate the update region. For example, if a window's position is switched from back to front, positions front to back, several changes occur in the regions of both windows. The system adds the lower-right corner of the new front window to that window's visible region. The system also adds that corner area to the window's update region.

Window Styles for Painting

Most of the styles relating to window drawing can be set either for the window class (CS_ prefix) or for an individual window (WS_ prefix). The styles described in this section control how the system manipulates the window's regions and how the window is notified when it must be painted or redrawn.

WS_CLIPCHILDREN, CS_CLIPCHILDREN

All the windows with this style are excluded from their parent's visible region. This style protects windows but increases the amount of time necessary to calculate the parent's visible region. This style normally is not necessary, because if the parent and child windows overlap and both are invalidated, the parent window is drawn before the child window. If the child window is invalidated independently from its parent window, only the child window is redrawn. If the update region of the parent window does not intersect the child window, drawing the parent

window does not disturb the child window.

WS_CLIPSIBLINGS, CS_CLIPSIBLINGS

Windows with this style are excluded from the visible region of sibling windows. This style protects windows with the same parent from being drawn accidentally, but increases the amount of time necessary to calculate the visible region. This style is appropriate for sibling windows that overlap.

WS_PARENTCLIP, CS_PARENTCLIP

The visible region for a window with this style is the same as the visible region of the parent window. This style simplifies the calculation of the visible region but is potentially hazardous, because the parent window's visible region usually is larger than the child window. Windows with this style should not draw outside their boundaries.

WS_SAVEBITS, CS_SAVEBITS

The system saves the bits beneath a window with this style when the window is displayed. When the window moves or is hidden, the system simply restores the uncovered bits. This operation can consume a great deal of memory; it is recommended only for transient windows such as menus and dialog boxes-not for main application windows. This style also is inappropriate for windows that are updated dynamically, such as clocks.

WS_SYNCPAINT, CS_SYNCPAINT

Windows that have these styles receive WM_PAINT messages as soon as their update regions contain something; they are updated immediately (synchronously).

CS_SIZEREDRAW

A window with this class style receives a WM_PAINT message; the window is completely invalidated whenever it is resized, even if it is made smaller. (Typically, only the uncovered area of a window is invalidated when a window is resized.) This class style is useful when an application scales graphics to fill the current window.

Strategies for Painting and Drawing

A PM application shares the screen with other windows and applications; therefore, painting and drawing must not interfere with those other applications and windows. When you follow these strategies, your application can coexist with other applications and still take full advantage of the graphics capabilities of the operating system.

Drawing in a Window

Ideally, all drawing in a window occurs as a result of an application's processing a WM_PAINT message. Applications maintain an internal representation of what must be displayed in the window, such as text or a linked list of graphics objects, and use the WM_PAINT message as a cue to display a visual representation of that data in the window.

To route all display output through the WM_PAINT message, an application must not draw on the screen at the time its data changes. Instead, it must update the internal representation of the data and call the WinInvalidateRect or WinInvalidateRegion functions to invalidate the portion of the window that must be redrawn. Sometimes it is much more efficient to draw directly in a window without relying on the WM_PAINT message—for example, when drawing and redrawing an object for a user who is using the mouse to drag or size the object.

If a window has the WS_SYNCPAINT or CS_SYNCPAINT style, invalidating a portion of the window causes a WM_PAINT message to be sent to the window immediately. Essentially, sending a message is like making a function call; the actions corresponding to the WM_PAINT message are carried out before the call that caused the invalidation returns—that is to say, the painting is synchronous.

If the window does not have the WS_SYNCPAINT or CS_SYNCPAINT style, invalidating a portion of the window causes the invalidated region to be added to the window's update region. The next time the application calls the WinGetMsg or WinPeekMsg functions, the application is sent a WM_PAINT message. If there are many messages in the queue, the painting occurs after the invalidation—that is, the painting is asynchronous. A WM_PAINT message is not posted to the queue in this case, so all invalidation operations since the last WM_PAINT message are consolidated into a single WM_PAINT message the next time the application has no messages in the queue.

There are advantages to both synchronous and asynchronous painting. Windows that have simple painting functions should be painted synchronously. Most of the system-defined control windows, such as buttons and frame controls, are painted synchronously because they can be painted quickly without interfering with the responsiveness of the program. Windows that require more time-consuming painting operations should be painted asynchronously so that the painting can be initiated only when there are no other pending messages that might otherwise be blocked while waiting for the window to be painted. Also, a window that uses an incremental approach to invalidating small portions of itself usually should allow those operations to consolidate into a single asynchronous WM_PAINT message, rather than a series of synchronous WM_PAINT messages. If necessary, an application can call the WinUpdateWindow function to cause an asynchronous window to update itself without going through the event loop. WinUpdateWindow sends a WM_PAINT message directly to the window if the window's update region is not empty.

The WM_PAINT Message

A window receives a WM_PAINT message whenever its update region is not NULL. A window procedure responds to a WM_PAINT message by calling the WinBeginPaint function, drawing to fill in the update areas, then calling the WinEndPaint function.

The WinBeginPaint function returns a handle to a presentation space that is associated with the device context for the window and that has a visible region equal to the intersection of the window's update region and its visible region. This means that only those portions of the window that need to be redrawn are drawn. Attempts to draw outside this region are clipped and do not appear on the screen.

If the application maintains its own presentation space for the window, it can pass the handle of that presentation space to WinBeginPaint, which modifies the visible region of the presentation space and passes the presentation-space handle back to the caller. If the application does not have its own presentation space, it can pass a NULL presentation-space handle and the system will return a cached-micro presentation space for the window. In either case, the application can use the presentation space to draw in the window.

The WinBeginPaint function takes a pointer to a RECT structure, filling in this structure with the coordinates of the rectangle that encloses the area to be updated. The application can use this rectangle to optimize drawing, by drawing only those portions of the window that intersect with the rectangle. If an application passes a NULL pointer for the rectangle argument, the application draws the entire window and relies on the clipping mechanism to filter out the unneeded areas.

After the WinBeginPaint function sets the update region of a window to NULL, the application does the necessary drawing to fill the update areas. If an application handles a WM_PAINT message and does not call WinBeginPaint, or otherwise empty the update region, the application continues to receive WM_PAINT messages as long as the update region is not empty.

After the application finishes drawing, it calls the WinEndPaint function to restore the presentation space to its former state. When a cached-micro presentation space is returned by WinBeginPaint, the presentation space is returned to the system for reuse. If the application supplies its own presentation space to WinBeginPaint, the presentation space is restored to its previous state.

Drawing the Minimized View

When an application creates a standard frame window, it has the option of specifying an icon that the system uses to represent the application in its minimized state. Typically, if an icon is supplied, the system draws it in the minimized window and labels it with the name of

the window. If the application does not specify the FS_ICON style for the window, the window receives a WM_PAINT message when it is minimized. The code in the window procedure that handles the WM_PAINT message can determine whether the frame window currently is minimized and draw accordingly. Notice that because the WS_MINIMIZED style is relevant only for the frame window, and not for the client window, the window procedure checks the frame window rather than the client window.

The following code fragment shows how to draw a window in both the minimized and normal states:

```
MRESULT EXPENTRY ClientWndProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
    HPS hps;
    RECTL rcl;
    ULONG flStyle;

    switch (msg) {
        case WM_PAINT:
            hps = WinBeginPaint(hwnd, (HPS) NULL, &rcl);

            /* Check whether the frame window
             (client's parent window)
             is minimized.                */

            flStyle = WinQueryWindowULong(WinQueryWindow(hwnd,
                QW_PARENT), QWL_STYLE);

            if (flStyle & WS_MINIMIZED) {
                .      /* Paint the minimized state. */
                .
            }
            else {
                .      /* Paint the normal state.      */
                .
            }
            WinEndPaint(hps);
            return 0;
    }
}
```

Drawing Without the WM_PAINT Message

An application can draw in a window's presentation space without having received a WM_PAINT message. As long as there is a presentation space for the window, an application can draw into the presentation space and avoid intruding into other windows or the desktop. Applications that draw without using the WM_PAINT message typically call the WinGetPS function to obtain a cached-micro presentation space for the window and call the WinReleasePS function when they have finished drawing. An application also can use any of the other types of presentation spaces described in the following sections.

Three Types of Presentation Spaces

All drawing must take place within a presentation space. The operating system provides three types of presentation spaces for drawing: normal, micro, and cached-micro presentation spaces. The *normal presentation space* provides the most functionality, allowing access to all the graphics functions of the operating system and enabling the application to draw to all device types. The normal presentation space is more difficult to use than the other two kinds of presentation spaces and it uses more memory. It is created by using the GpiCreatePS function and is destroyed by using the GpiDestroyPS function.

The *micro presentation space* allows access to only a subset of the operating system graphics functions, but it uses less memory and is faster than a normal presentation space. The micro presentation space also enables the application to draw to all device types. It is created by using the GpiCreatePS function and destroyed by using the GpiDestroyPS function.

The *cached-micro presentation space* provides the least functionality of the three kinds of presentation spaces, but it is the most efficient and easiest to use. The cached-micro presentation space draws only to the screen. It is created and destroyed by using either the WinBeginPaint and WinEndPaint functions or the WinGetPS and WinReleasePS functions.

The following sections describe each of the types of presentation spaces, in detail, and suggest strategies for using each type in an application. All three kinds of presentation spaces can be used in a single application. Some windows, especially if they never will be printed, are best served by cached-micro presentation spaces. Other windows might require the more flexible services of micro or normal presentation spaces.

Normal Presentation Spaces

The normal presentation space supports the full power of the operating system graphics, including retained graphics. The primary advantages of a normal presentation space over the other two presentation-space types are its support of all graphics functions and its ability to be associated with many kinds of device contexts.

A normal presentation space can be associated with many different device contexts. Typically, this means that an application creates a normal presentation space and associates it with a window device context for screen display. When the user asks to print, the application associates the same presentation space with a printer device context. Later, the application can reassociate the presentation space with the window device context. A presentation space can be associated with only one device context at a time, but the normal presentation space enables the application to change the device context whenever necessary.

When creating a normal presentation space, an application can associate it with a device context or defer the association to a later time. The GpiAssociate function associates a device context with a normal presentation space after the presentation space has been created. An application typically associates the normal presentation space with a device context when calling the GpiCreatePS function and, later, associates the presentation space with a different device context by calling GpiAssociate. To obtain a device context for a window, call the WinOpenWindowDC function. To obtain a device context for a device other than the screen, call the DevOpenDC function.

An application typically creates a normal presentation space during initialization and uses it until termination. Each time the application receives a WM_PAINT message, it passes the handle of the normal presentation space as an argument to WinBeginPaint; this prevents the system from returning a cached-micro presentation space. The system modifies the visible region of the supplied normal presentation space and returns the presentation space to the application. This method enables the application to use the same presentation space for all the drawing in a specified window. Normal presentation spaces created using GpiCreatePS must be destroyed by calling GpiDestroyPS before the application terminates. Do not call WinReleasePS to release a presentation space obtained using GpiCreatePS. Before terminating, applications also must use DevCloseDC to close any device contexts opened using DevOpenDC. No action is necessary for device contexts obtained using WinOpenWindowDC, because the system automatically closes these device contexts when destroying the associated windows.

Micro Presentation Spaces

The primary advantage of a micro presentation space over a cached-micro presentation space is that it can be used for printing as well as painting in a window. An application that uses a micro presentation space must explicitly associate it with a device context. This makes the micro presentation space useful for painting to a printer, a plotter, or an off-screen memory bit map.

A micro presentation space does not support the full set of OS/2 graphics functions. Unlike a normal presentation space, a micro presentation space does not support retained graphics.

An application that must display graphics or text in a window and print to a printer or plotter typically maintains two presentation spaces: one for the window and one for the printing device. The following figure shows how an application's graphics output can be routed through separate presentation spaces to produce a screen display and printed copy. An application creates a micro presentation space by calling the GpiCreatePS function. A device context must be supplied at the time the micro presentation space is created. An application typically creates a device context and then a presentation space. The following code fragment demonstrates this by obtaining a device context for a window and associating it with a new micro presentation space:

```
hdc = WinOpenWindowDC(...);  
hps = GpiCreatePS(..., hdc, ..., GPIA_ASSOC);
```

To create a micro presentation space for a device other than the screen, replace the call to the WinOpenWindowDC function with a call to the DevOpenDC function, which obtains a device context for a device other than the screen. Then the device context that is obtained by this

call can be used as an argument to GpiCreatePS.

An application typically creates a micro presentation space during initialization and uses it until termination. Each time the application receives a WM_PAINT message, it should pass the handle of the micro presentation space as an argument to the WinBeginPaint function; this prevents the system from returning a cached-micro presentation space. The system modifies the visible region of the supplied micro presentation space and returns the presentation space to the application. This method enables the application to use the same presentation space for all drawing in a specified window.

Micro presentation spaces created by using GpiCreatePS should be destroyed by calling GpiDestroyPS before the application terminates. Do not call the WinReleasePS function to release a presentation space obtained by using GpiCreatePS. Before terminating, applications must use the DevCloseDC function to close any device contexts opened using the DevOpenDC function. No action is necessary for device contexts obtained using WinOpenWindowDC, because the system automatically closes these device contexts when destroying the associated windows.

Cached-Micro Presentation Spaces

The cached-micro presentation space provides the simplest and most efficient drawing environment. It can be used only for drawing on the screen, typically in the context of a window. It is most appropriate for application tasks that require simple window-drawing functions that will not be printed. Cached-micro presentation spaces do not support retained graphics.

After an application draws to a cached-micro presentation space, the drawing commands are routed through an implied device context to the current display. The application does not require information about the actual device context, because the device context is assumed to be the display. This process makes cached-micro presentation spaces easy for applications to use. The following code fragment illustrates this process:

```
HPS    hps;

case WM_PAINT:
    hps = WinBeginPaint(hwnd,NULL,NULL);

    /*
     * Use PS.
     */

    WinEndPaint (hps);
```

or

```
HPS    hps;

case WM_PAINT:

    hps = WinGetPS(hwnd);

    /*
     * Use PS.
     */

    WinReleasePS(hps);
```

There are two common strategies for using cached-micro presentation spaces in an application. The simplest strategy is to call the WinBeginPaint function during the WM_PAINT message, use the resulting cached-micro presentation space to draw in the window, then return the presentation space to the system by calling the WinEndPaint function. By using this method, the application interacts with the presentation space only when drawing in the presentation space. This method is most appropriate for simple drawing. A disadvantage of this method is that the application must set up any special attributes for the presentation space, such as line color and font, each time a new presentation space is obtained.

A second strategy is for the application to allocate a cached-micro presentation space during initialization, by calling the WinGetPS function and saving the resulting presentation-space handle in a static variable. Then the application can set attributes in the presentation space that exist for the life of the program. The presentation-space handle can be used as an argument to the WinBeginPaint function each time the window gets a WM_PAINT message; the system modifies the visible region and returns the presentation space to the application with its attributes intact. This strategy is appropriate for applications that need to customize their window-drawing attributes. A presentation space that is obtained by calling the WinGetPS function must be released by calling WinReleasePS when the application has finished using it, typically during program termination. A presentation space that is obtained by calling WinBeginPaint must be released by calling WinEndPaint, typically as the last part of processing a WM_PAINT message.

Presentation Parameters

A presentation parameter is a piece of data that can be attached to a window to influence the way it is painted. An application can use system-defined presentation parameters to change the colors used by the various PM controls. It can also use a presentation parameter to change the font used for text. If you want to create your own presentation parameters, you must use indexes above PP_USER.

About Presentation Parameters

An application can attach any presentation parameter to a window. However, the window's implementation determines whether or not the presentation parameter will have an effect on the window. For example, if a window does not contain any text, the window will ignore the setting of a font presentation parameter. In general, a well-behaved window should support the more commonly used presentation parameters, such as those for setting the foreground and background colors and the text font. OS/2 uses presentation parameters to let users change window fonts and colors by dragging colors and fonts from Workplace Shell palettes and dropping them in the window. If a window does not support the correct presentation parameters, it can appear unresponsive to the user's actions.

By default, presentation parameter values are inherited from the window's owner. Suppose you create a dialog window that has radio buttons. You don't have to set the same background color in both the dialog window and the radio buttons because the radio button will inherit its background color from the dialog window. Any time the dialog's background color is changed, the radio button's color changes automatically.

WinSetPresParam is used to attach a presentation parameter to a window. Although any presentation parameter index can be used, windows typically support only a small subset of the possible values. For example, the Presentation Manager controls support the PP_FONTNAMESIZE presentation parameter and several of the color and color index presentation parameters. Setting a supported presentation parameter to a window will cause the window to repaint itself, making use of the new value.

Setting a color presentation parameter to a window does not automatically change a color. It is up to the window's implementation to query the presentation parameter's value, and then use it when updating the window.

Querying and Setting Presentation Parameters

To implement support for a presentation parameter in a window, you need to find out what its value is and when that value has been changed. To query a presentation parameter's value use WinQueryPresParam, passing in the index of the presentation parameter and a buffer large enough for the value to be returned in. For presentation parameters that have string values, the buffer should be large enough to include the null termination character.

The flag QPF_NOINHERIT can be specified if the presentation parameter should not be inherited from the window's owner. Otherwise the system looks up the window's owner chain until it finds an occurrence of the same presentation parameter index.

When a presentation parameter is set or changed, the window receives a WM_PRESPARAMCHANGED message. If only one value is changed, that presentation parameter's index is passed with the message. If more than one value is changed, zero is passed. In the latter case, the window must query all its presentation parameters to find out which ones have changed. In response to the WM_PRESPARAMCHANGED message, the receiving window should repaint itself according to the new values received. It is not possible to reject a presentation parameter change; the new value has already been set by the time the notification message has been received.

WinRemovePresParam is available for removing a presentation parameter from a window.

```
/* ***** */
/* Set the background color of the window */
```

```

/*****
lColor = RGB_RED;                                /* RGB color value */
WinSetPresParam(hwndMain,                        /* window handle */
    PP_BACKGROUNDCOLOR,                          /* background pres param */
    sizeof(lColor),                             /* length of pres param */
    &lColor);                                    /* pres param value */

-----

... in window procedure for hwndMain ...

case WM_PRESPARAMCHANGED:

    /*****
    /* Message received when a presparam has been changed. */
    /* Invalidate the window to repaint with new color. */
    /*****

    WinInvalidateRect(hwndMain, NULL, FALSE);
    break;

case WM_PAINT:

    /*****
    /* Repaint the window */
    /*****

    hps = WinBeginPaint(hwndMain, NULLHANDLE, NULL);

    /*****
    /* Put presentation space into RGB mode */
    /*****

    GpiCreateLogColorTable(hps, 0, LCOLF_RGB, 0, 0, NULL);

    /*****
    /* Query presentation parameter value for background color */
    /*****

    WinQueryPresParam(hwndMain,                /* Window handle */
        PP_BACKGROUNDCOLOR,                  /* Background presparam */
        0,
        NULL,
        sizeof(lColor),                      /* Length of data buffer */
        &lColor,                             /* Data buffer returned */
        0);

    /*****
    /* Fill window with background color retrieved from presparam */
    /*****

    WinQueryWindowRect(hwndMain, &rectMain);
    WinFillRect(hps, &rectMain, lColor);

    ... rest of painting code ...

    WinEndPaint(hps);
    break;

```

Creating a Window with a Presentation Parameter

It is possible to pass in presentation parameters when creating a window, using `WinCreateWindow`. The last parameter *pPresParams* is a pointer to a `PRESPARAMS` structure containing an array of `PARAM` structures, which in turn contain the presentation parameter indexes and values. The size field in `PRESPARAMS` should be set to the size of the array being passed.

```

/*****

```

```

/* Set a presentation parameter when creating a window */
/*****

PPRESPARAMS ppresMain;

lColor = RGB_RED;

/*****
/* Allocate space for PRESPARAMS structure and one presparam */
/*****

ppresMain = (PPRESPARAMS)malloc(sizeof(ULONG) * 4);

/*****
/* Set up PRESPARAMS structure with a background color */
/*****

ppresMain->cb = sizeof(ULONG) * 3;
ppresMain->aparam[0].id = PP_BACKGROUNDCOLOR;
ppresMain->aparam[0].cb = sizeof(lColor);

memcpy(&ppresMain->aparam[0].ab, &lColor, sizeof(lColor));

/*****
/* Create the window and pass in the background color presparam */
/*****

hwndMain = WinCreateWindow(hwndFrame,
                           "MainWindow",
                           0,
                           0,
                           0, 0, 0, 0,
                           0,
                           HWND_TOP,
                           FID_CLIENT,
                           NULL,
                           ppresMain);    /* Pass presparam data */

```

Specifying PRESPARAMS in a Dialog Template

If you are creating a dialog window, you can specify presentation parameters inside the dialog template. For any window in a dialog template (including the dialog window itself) you may add one or more PRESPARAMS statements to the template following the definition of the control.

Note: For the dialog window, place the PRESPARAMS statement immediately after the DIALOG statement in the template.

```

DLGTEMPLATE DLG_MYDIALOG
BEGIN
    DIALOG "Add Expression", DLG_MYDIALOG, 30, 25, 205, 55,
        WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
    /* Set font to be used in dialog */
    PRESPARAMS PP_FONTNAMESIZE, "10.Helv"
    BEGIN
        LTEXT "Enter new expression:", -1, 5, 40, 195, 8
        /* Set above static control to yellow text on red background */
        PRESPARAMS PP_BACKGROUNDCOLOR, RGB_RED
        PRESPARAMS PP_FOREGROUNDINDEX, CLR_YELLOW
        ENTRYFIELD "", -1, 7, 27, 191, 8, ES_MARGIN
        PUSHBUTTON "OK", DID_OK, 5, 5, 45, 14
    END
END

```

Querying Color Presentation Parameter Pairs

Many of the color presentation parameters come in pairs. For example, the background color of a window can be set using either PP_BACKGROUND_COLOR, which takes an RGB color value (for example, RGB_RED), or PP_BACKGROUND_COLOR_INDEX, which takes a color index (for example, CLR_RED). The newer presentation parameters do not have a color index equivalent, so it is recommended that you use RGB values wherever possible. RGB presentation parameters will also work with SYSCLR_ index values and negative CLR_ indexes (for example, CLR_BLACK).

WinQueryPresParam can be used to query a pair of color presentation parameters at once. The first parameter takes precedence over the second, and should normally be the RGB presentation parameter. The color index should be the second. If you are querying a color index presentation parameter and you want the color value to be returned as an RGB value, you can specify either QPF_ID1COLORINDEX or QPF_ID2COLORINDEX to convert a color index value to an RGB value.

If a window does not have a presentation parameter set for a particular color, the window must select a default color to use. For example, if there is no value set for PP_FOREGROUND_COLOR or PP_FOREGROUND_COLOR_INDEX, the window has to choose a default foreground color when it paints. The usual practice is to select a corresponding system color (for example, SYSCLR_WINDOWTEXT) and use that to paint the foreground of the window.

The decision of which system color to use is an important one, as system colors can be changed by the user with a Scheme Palette. Your choice should always be appropriate to the functionality of the window. If the user does change a system color, your window will receive a WM_SYSCOLORCHANGED message to notify you that your default colors might have been changed.

If there is no appropriate system color available, you will need to select a hard-coded color value as the default instead.

```

/*****
/* Query presentation parameter value for background color */
*****/

if (WinQueryPresParam(
    hwndMain,          /* Window handle */
    PP_BACKGROUND_COLOR, /* Background presparam (RGB) */
    PP_BACKGROUND_COLOR_INDEX, /* Background presparam (Index) */
    NULL,
    sizeof(lColor),    /* Length of data buffer */
    &lColor,            /* Data buffer returned */
    QPF_ID2COLORINDEX) == 0) /* Convert 2nd presparam to RGB */
{
    /*****
    /* No presparam found - query default background color */
    *****/

    lColor = WinQuerySysColors(
        HWND_DESKTOP, /* Desktop window handle */
        SYSCLR_BACKGROUND, /* System default background */
        0);            /* Reserved */
}

GpiSetBackColor(hps, lColor);

```

Setting a Font Presentation Parameter

There is one font presentation parameter, PP_FONTNAMESIZE, which is used to select the default font for a window. If this presentation parameter is attached to a window, the specified font is automatically selected as the default and is the one used when text-drawing PM functions are called (for example, WinDrawText and GpiCharStringAt).

The format of the font name string is as follows:

<point size>.<face name>(<modifier>(<modifier> ...

where:

<point size> is the point size of the font

<face name> is the face name of the font

<modifier> is one of the following:

- Bold
- Italic
- Underscore
- Outline
- Strikeout

Examples are "12.New Times Roman" and "10.Helvetica.Bold.Italic"

Note: Modifiers can be used to create a font with a combination of the listed attributes. Do not use modifiers if a true font of that type is already available. For example, use "10.Helvetica Bold" instead of "10.Helvetica.Bold."

```

/*****
/* Set the text font for the window */
*****/

szFont = "10.Helvetica Bold Italic";

WinSetPresParam(hwndMain,          /* window handle */
                PP_FONTNAMESIZE,    /* background pres param */
                strlen(szFont) + 1, /* length of pres param */
                szFont);            /* pres param value */

```

Colors Used by PM Controls

PM controls make extensive use of presentation parameters for determining the fonts and colors to paint with. All the controls that display text strings use the font set by the PP_FONTNAMESIZE presentation parameter. The colors used in the controls can be set using presentation parameters. Each control responds to a subset of the color presentation parameters, depending on the type of painting it is required to do. Some of the color presentation parameters are specific to a single control; others are used by many or all of the controls.

When there are no presentation parameters set, a control selects the default colors to use. If nothing else is available, the control will use the default system colors (obtained by calling WinQuerySysColor). When no system color is available, it will use a hard-coded RGB color.

In summary, the colors a control window uses depends on what has been set by the application and the user. The order of precedence (from first to last) for a PM control's choice of colors is as follows:

1. Presentation parameters
2. Application-specific control colors
3. Global control colors
4. Default colors (system colors, where available)

For a list of the default colors used by a system PM control, select its name from the following list:

[Static Bitmap](#)
[Static Text](#)
[Group Box](#)
[Push Button](#)
[Check Box](#)
[Radio Button](#)
[Entry Field](#)
[List Box](#)
[Combo Box](#)
[Title Bar](#)
[Menu](#)
[Frame](#)
[Scroll Bar](#)
[Spin Button](#)
[Slider](#)
[Circular Slider](#)
[MLE](#)
[Value Set](#)
[Notebook](#)
[Container](#)

Static Bitmap : CCT_STATIC

- Color Usage Description
 - Background
 - Background (in dialog)
 - Foreground
 - Disabled background

- Background color

Control Color Index	CCI_BACKGROUND
Presentation Parameter	PP_BACKGROUNDCOLOR
System Default	SYSCLR_WINDOW
Remarks	None

- Background color (in dialog)

Control Color Index	CCI_BACKGROUNDDDIALOG
Presentation Parameter	PP_BACKGROUNDCOLOR
System Default	SYSCLR_DIALOGBACKGROUND
Remarks	None

- Foreground color

Control Color Index	CCI_FOREGROUND
Presentation Parameter	PP_FOREGROUNDCOLOR
System Default	SYSCLR_WINDOWFRAME
Remarks	None

- Disabled background color

Control Color Index	CCI_DISABLEDBACKGROUND
Presentation Parameter	PP_DISABLEDBACKGROUNDCOLOR
System Default	SYSCLR_BACKGROUND
Remarks	None

Static Text : CCT_STATICTEXT

- Color Usage Description
 - Background
 - Background (in dialog)
 - Text

- Background color

Control Color Index	CCI_BACKGROUND
Presentation Parameter	PP_BACKGROUNDCOLOR
System Default	SYSCLR_WINDOW
Remarks	None

- Background color (in dialog)

Control Color Index	CCI_BACKGROUNDDDIALOG
Presentation Parameter	PP_BACKGROUNDCOLOR
System Default	SYSCLR_DIALOGBACKGROUND
Remarks	None

- Text color

Control Color Index	CCI_FOREGROUND
Presentation Parameter	PP_FOREGROUNDCOLOR
System Default	SYSCLR_WINDOWSTATICTEXT
Remarks	None

Group Box : CCT_GROUPBOX

- Color Usage Description
 - Background
 - Background (in dialog)
 - Text
 - Light Border
 - Dark Border

- Background color

Control Color Index	CCI_BACKGROUND
Presentation Parameter	PP_BACKGROUNDCOLOR
System Default	SYSCLR_WINDOW

Remarks	Not inherited
• Background color (in dialog)	
Control Color Index	CCI_BACKGROUNDDDIALOG
Presentation Parameter	PP_BACKGROUNDCOLOR
System Default	SYSCLR_DIALOGBACKGROUND
Remarks	Not inherited
• Text color	
Control Color Index	CCI_FOREGROUND
Presentation Parameter	PP_FOREGROUNDCOLOR
System Default	SYSCLR_WINDOWSTATICTEXT
Remarks	Not inherited
• Light border color	
Control Color Index	CCI_BORDERLIGHT
Presentation Parameter	PP_BORDERLIGHTCOLOR
System Default	RGB_WHITE
Remarks	Not inherited
• Dark border color	
Control Color Index	CCI_BORDERDARK
Presentation Parameter	PP_BORDERDARKCOLOR
System Default	RGB_DARKGRAY
Remarks	RGB_DARKGRAY not defined in OS/2 header files. Its value is 0x00808080.

Push Button: CCT_PUSHBUTTON

- Color Usage Description
 - Background
 - Text
 - Disabled background
 - Disabled text
 - Default border

Light border
Dark border

- Background color

Control Color Index	CCI_BACKGROUND
Presentation Parameter	PP_BACKGROUND_COLOR
System Default	SYSCLR_BUTTONMIDDLE
Remarks	Not inherited.

- Text color

Control Color Index	CCI_FOREGROUND
Presentation Parameter	PP_FOREGROUND_COLOR
System Default	SYSCLR_MENUTEXT
Remarks	Not inherited.

- Disabled background color

Control Color Index	CCI_DISABLEDBACKGROUND
Presentation Parameter	PP_DISABLEDBACKGROUND_COLOR
System Default	SYSCLR_BUTTONMIDDLE
Remarks	Not inherited.

- Disabled text color

Control Color Index	CCI_DISABLEDFOREGROUND
Presentation Parameter	PP_DISABLEDFOREGROUND_COLOR
System Default	SYSCLR_MENUTEXT
Remarks	Not inherited.

- Default Border color

Control Color Index	CCI_BORDERDEFAULT
Presentation Parameter	PP_BORDERDEFAULT_COLOR
System Default	SYSCLR_BUTTONDEFAULT
Remarks	None

- Light border color

Control Color Index	CCI_BORDERLIGHT
Presentation Parameter	PP_BORDERLIGHT_COLOR
System Default	SYSCLR_BUTTONLIGHT

Remarks	None
---------	------

- Dark border color

Control Color Index	CCI_BORDERDARK
Presentation Parameter	PP_BORDERDARKCOLOR
System Default	SYSCLR_BUTTONDARK
Remarks	None

Check Box : CCT_CHECKBOX

- Color Usage Description

Background

Background (in dialog)

Text

Disabled background

Disabled background (in dialog)

Disabled text

Highlight background

Highlight background (in dialog)

Highlight text
- Background color

Control Color Index	CCI_BACKGROUND
Presentation Parameter	PP_BACKGROUNDCOLOR
System Default	SYSCLR_BUTTONMIDDLE
Remarks	None
- Background (in dialog) color

Control Color Index	CCI_BACKGROUNDDDIALOG
Presentation Parameter	PP_BACKGROUNDCOLOR
System Default	SYSCLR_DIALOGBACKGROUND
Remarks	None
- Text color

Control Color Index	CCI_FOREGROUND
Presentation Parameter	PP_FOREGROUNDCOLOR
System Default	SYSCLR_MENUTEXT

Remarks	None
• Disabled background color	
Control Color Index	CCI_DISABLEDBACKGROUND
Presentation Parameter	PP_DISABLEDBACKGROUNDCOLOR
System Default	SYSCLR_WINDOW
Remarks	None
• Disabled background (in dialog) color	
Control Color Index	CCI_DISABLEDBACKGROUNDDDIALOG
Presentation Parameter	PP_DISABLEDBACKGROUNDDCOLOR
System Default	SYSCLR_DIALOGBACKGROUND
Remarks	None
• Disabled text color	
Control Color Index	CCI_DISABLEDFOREGROUND
Presentation Parameter	PP_DISABLEDFOREGROUNDCCOLOR
System Default	SYSCLR_MENUTEXT
Remarks	None
• Highlight background color	
Control Color Index	CCI_HIGHLIGHTBACKGROUND
Presentation Parameter	PP_HILITEBACKGROUNDCCOLOR
System Default	SYSCLR_WINDOW
Remarks	None
• Highlight background (in dialog) color	
Control Color Index	CCI_HIGHLIGHTBACKGROUNDDDIALOG
Presentation Parameter	PP_HILITEBACKGROUNDDCOLOR
System Default	SYSCLR_DIALOGBACKGROUND
Remarks	None
• Highlight text color	
Control Color Index	CCI_HIGHLIGHTFOREGROUND
Presentation Parameter	PP_HILITEFOREGROUNDCCOLOR

System Default	SYSCLR_MENUTEXT
Remarks	None

Radio Button : CCT_RADIOBUTTON

- Color Usage Description

Background
Background (in dialog)
Text
Disabled background
Disabled background (in dialog)
Disabled text
Highlight background
Highlight background (in dialog)
Highlight text

- Background color

Control Color Index	CCI_BACKGROUND
Presentation Parameter	PP_BACKGROUNDCOLOR
System Default	SYSCLR_BUTTONMIDDLE
Remarks	None

- Background (in dialog) color

Control Color Index	CCI_BACKGROUNDDDIALOG
Presentation Parameter	PP_BACKGROUNDCOLOR
System Default	SYSCLR_DIALOGBACKGROUND
Remarks	None

- Text color

Control Color Index	CCI_FOREGROUND
Presentation Parameter	PP_FOREGROUNDCOLOR
System Default	SYSCLR_MENUTEXT
Remarks	None

- Disabled background color

Control Color Index	CCI_DISABLEDBACKGROUND
Presentation Parameter	PP_DISABLEDBACKGROUNDCOLOR
System Default	SYSCLR_WINDOW

Remarks	None
• Disabled background (in dialog) color	
Control Color Index	CCI_DISABLEDBACKGROUNDDDIALOG
Presentation Parameter	PP_DISABLEDBACKGROUNDDCOLOR
System Default	SYSCLR_DIALOGBACKGROUND
Remarks	None
• Disabled text color	
Control Color Index	CCI_DISABLEDFOREGROUND
Presentation Parameter	PP_DISABLEDFOREGROUNDCCOLOR
System Default	SYSCLR_MENUTEXT
Remarks	None
• Highlight background color	
Control Color Index	CCI_HIGHLIGHTBACKGROUND
Presentation Parameter	PP_HILITEBACKGROUNDCCOLOR
System Default	SYSCLR_WINDOW
Remarks	None
• Highlight background (in dialog) color	
Control Color Index	CCI_HIGHLIGHTBACKGROUNDDDIALOG
Presentation Parameter	PP_HILITEBACKGROUNDCCOLOR
System Default	SYSCLR_DIALOGBACKGROUND
Remarks	None
• Highlight text color	
Control Color Index	CCI_HIGHLIGHTFOREGROUND
Presentation Parameter	PP_HILITEFOREGROUNDCCOLOR
System Default	SYSCLR_MENUTEXT
Remarks	None

Entry Field : CCT_ENTRYFIELD

- Color Usage Description

Background
Text
Read-only text
Disabled background
Disabled text
Disabled read-only text
Highlight background
Highlight text
Light border
Dark border
Light border 2
Dark border 2

- Background color

Control Color Index	CCI_BACKGROUND
Presentation Parameter	PP_BACKGROUNDDCOLOR
System Default	SYSCLR_ENTRYFIELD
Remarks	Not inherited.

- Text color

Control Color Index	CCI_FOREGROUND
Presentation Parameter	PP_FOREGROUNDDCOLOR
System Default	SYSCLR_WINDOWTEXT
Remarks	Pure RGB color.

- Read-only text color

Control Color Index	CCI_FOREGROUNDREADONLY
Presentation Parameter	PP_FOREGROUNDDCOLOR
System Default	SYSCLR_OUTPUTTEXT
Remarks	Pure RGB color.

- Disabled background color

Control Color Index	CCI_DISABLEDBACKGROUND
Presentation Parameter	PP_DISABLEDBACKGROUNDDCOLOR
System Default	SYSCLR_ENTRYFIELD
Remarks	Not inherited

- Disabled text color

Control Color Index	CCI_DISABLEDFOREGROUND
---------------------	------------------------

	Presentation Parameter	PP_DISABLEDFOREGROUND
	System Default	SYSCLR_WINDOWTEXT
	Remarks	Pure RGB color.
•	Disabled read-only text color	
	Control Color Index	CCI_DISABLEDFOREGROUNDREADONLY
	Presentation Parameter	PP_DISABLEDFOREGROUND
	System Default	SYSCLR_OUTPUTTEXT
	Remarks	Pure RGB color.
•	Highlight background color	
	Control Color Index	CCI_HIGHLIGHTBACKGROUND
	Presentation Parameter	PP_HILITEBACKGROUND
	System Default	SYSCLR_HILITEBACKGROUND
	Remarks	None
•	Highlight text color	
	Control Color Index	CCI_HIGHLIGHTFOREGROUND
	Presentation Parameter	PP_HILITEFOREGROUND
	System Default	SYSCLR_HILITEFOREGROUND
	Remarks	Pure RGB color.
•	Light border color	
	Control Color Index	CCI_BORDERLIGHT
	Presentation Parameter	PP_BORDERLIGHTCOLOR
	System Default	RGB_WHITE
	Remarks	None
•	Dark border color	
	Control Color Index	CCI_BORDERDARK
	Presentation Parameter	PP_BORDERDARKCOLOR
	System Default	RGB_DARKGRAY
	Remarks	RGB_DARKGRAY not defined in OS/2 header files. Its value is 0x00808080.
•	Light border 2 color	

Control Color Index	CCI_BORDER2LIGHT
Presentation Parameter	PP_BORDER2LIGHTCOLOR
System Default	RGB_LIGHTGRAY
Remarks	RGB_LIGHTGRAY not defined in OS/2 header files. Its value is 0x00cccccc.

- Dark border 2 color

Control Color Index	CCI_BORDER2DARK
Presentation Parameter	PP_BORDER2DARKCOLOR
System Default	RGB_BLACK
Remarks	None

List Box : CCT_LISTBOX

- Color Usage Description

Background
Text
Disabled background
Disabled text
Highlight background
Highlight text
Light border
Dark border
Light border 2
Dark border 2
Corner between scroll bars

- Background color

Control Color Index	CCI_BACKGROUND
Presentation Parameter	PP_BACKGROUNDCOLOR
System Default	SYSCLR_ENTRYFIELD
Remarks	Not inherited.

- Text color

Control Color Index	CCI_FOREGROUND
Presentation Parameter	PP_FOREGROUNDCOLOR
System Default	SYSCLR_WINDOWTEXT
Remarks	None

- Disabled background color

Control Color Index	CCI_DISABLEDBACKGROUND
Presentation Parameter	PP_DISABLEDBACKGROUNDCOLOR
System Default	SYSCLR_ENTRYFIELD
Remarks	Not inherited

- Disabled text color

Control Color Index	CCI_DISABLEDFOREGROUND
Presentation Parameter	PP_DISABLEDFOREGROUNDCOLOR
System Default	SYSCLR_WINDOWTEXT
Remarks	None

- Highlight background color

Control Color Index	CCI_HIGHLIGHTBACKGROUND
Presentation Parameter	PP_HILITEBACKGROUNDCOLOR
System Default	SYSCLR_HILITEBACKGROUND
Remarks	None

- Highlight text color

Control Color Index	CCI_HIGHLIGHTFOREGROUND
Presentation Parameter	PP_HILITEFOREGROUNDCOLOR
System Default	SYSCLR_HILITEFOREGROUND
Remarks	Pure RGB color.

- Light border color

Control Color Index	CCI_BORDERLIGHT
Presentation Parameter	PP_BORDERLIGHTCOLOR
System Default	RGB_WHITE
Remarks	None

- Dark border color

Control Color Index	CCI_BORDERDARK
Presentation Parameter	PP_BORDERDARKCOLOR
System Default	RGB_DARKGRAY

Remarks	RGB_DARKGRAY not defined in OS/2 header files. Its value is 0x00808080.
• Light border 2 color	
Control Color Index	CCI_BORDER2LIGHT
Presentation Parameter	PP_BORDER2LIGHTCOLOR
System Default	RGB_LIGHTGRAY
Remarks	RGB_LIGHTGRAY not defined in OS/2 header files. Its value is 0x00cccccc.
• Dark border 2 color	
Control Color Index	CCI_BORDER2DARK
Presentation Parameter	PP_BORDER2DARKCOLOR
System Default	RGB_BLACK
Remarks	None
• Corner between scroll bars color	
Control Color Index	CCI_FIELDBACKGROUND
Presentation Parameter	PP_FIELDBACKGROUNDCOLOR
System Default	SYSCLR_FIELDBACKGROUND
Remarks	None

Combo Box : CCT_COMBOBOX

• Color Usage Description	Background Text Light border Dark border Light border 2 Dark border 2 Button background Button light border Button dark border Arrow
• Background color	Control Color Index
	CCI_BACKGROUND

	Presentation Parameter	PP_BACKGROUND_COLOR
	System Default	SYSCLR_ENTRYFIELD
	Remarks	None
•	Text color	
	Control Color Index	CCI_FOREGROUND
	Presentation Parameter	PP_FOREGROUND_COLOR
	System Default	SYSCLR_WINDOWTEXT
	Remarks	Not inherited
•	Light border color	
	Control Color Index	CCI_BORDERLIGHT
	Presentation Parameter	PP_BORDERLIGHT_COLOR
	System Default	RGB_WHITE
	Remarks	None
•	Dark border color	
	Control Color Index	CCI_BORDERDARK
	Presentation Parameter	PP_BORDERDARK_COLOR
	System Default	RGB_DARKGRAY
	Remarks	RGB_DARKGRAY not defined in OS/2 header files. Its value is 0x00808080.
•	Light border 2 color	
	Control Color Index	CCI_BORDER2LIGHT
	Presentation Parameter	PP_BORDER2LIGHT_COLOR
	System Default	RGB_LIGHTGRAY
	Remarks	RGB_LIGHTGRAY not defined in OS/2 header files. Its value is 0x00cccccc.
•	Dark border 2 color	
	Control Color Index	CCI_BORDER2DARK
	Presentation Parameter	PP_BORDER2DARK_COLOR
	System Default	RGB_BLACK
	Remarks	None

- Button background color

Control Color Index	CCI_BUTTONBACKGROUND
Presentation Parameter	PP_BUTTONBACKGROUND
System Default	SYSCLR_BUTTONMIDDLE
Remarks	None

- Button light border color

Control Color Index	CCI_BUTTONBORDERLIGHT
Presentation Parameter	PP_BUTTONBORDERLIGHTCOLOR
System Default	SYSCLR_BUTTONLIGHT
Remarks	None

- Button dark border color

Control Color Index	CCI_BUTTONBORDERDARK
Presentation Parameter	PP_BUTTONBORDERDARKCOLOR
System Default	SYSCLR_BUTTONDARK
Remarks	None

- Arrow color

Control Color Index	CCI_ARROW
Presentation Parameter	PP_ARROWCOLOR
System Default	RGB_BLACK
Remarks	None

Title Bar : CCT_TITLEBAR

- Color Usage Description

Inactive text
Inactive background
Inactive text background
Active text
Active background
Active text background
Light border
Dark border

- Inactive text color

Control Color Index	CCI_INACTIVEFOREGROUND
Presentation Parameter	PP_INACTIVETEXTFGNDGROUNDCOLOR
System Default	SYSCLR_INACTIVETITLETEXT
Remarks	Not inherited.

- Inactive background color

Control Color Index	CCI_INACTIVEBACKGROUND
Presentation Parameter	PP_INACTIVECOLOR
System Default	SYSCLR_INACTIVETITLE
Remarks	Not inherited.

- Inactive text background color

Control Color Index	CCI_INACTIVEBACKGROUNDTEXT
Presentation Parameter	PP_INACTIVETEXTBGNDGROUNDCOLOR
System Default	SYSCLR_INACTIVETITLETEXT
Remarks	Not inherited.

- Active text color

Control Color Index	CCI_ACTIVEFOREGROUND
Presentation Parameter	PP_ACTIVETEXTFGNDGROUNDCOLOR
System Default	SYSCLR_ACTIVETITLETEXT
Remarks	Not inherited.

- Active background color

Control Color Index	CCI_ACTIVEBACKGROUND
Presentation Parameter	PP_ACTIVECOLOR
System Default	SYSCLR_ACTIVETITLE
Remarks	Not inherited.

- Active text background color

Control Color Index	CCI_ACTIVEBACKGROUNDTEXT
Presentation Parameter	PP_ACTIVETEXTBGNDGROUNDCOLOR
System Default	SYSCLR_ACTIVETITLETEXTBGND
Remarks	Not inherited.

- Light border color

Control Color Index	CCI_BORDERLIGHT
Presentation Parameter	PP_BORDERLIGHTCOLOR
System Default	RGB_WHITE
Remarks	None

- Dark border color

Control Color Index	CCI_BORDERDARK
Presentation Parameter	PP_BORDERDARKCOLOR
System Default	RGB_DARKGRAY
Remarks	RGB_DARKGRAY not defined in OS/2 header files. Its value is 0x00808080.

Menu : CCT_MENU

- Color Usage Description

Background
 Text
 Disabled background
 Disabled text
 Highlight background
 Highlight text
 Light border
 Dark border
 Light border 2
 Dark border 2
 Arrow
 Light arrow border
 Dark arrow border
 Light check mark
 Middle check mark
 Dark check mark

- Background color

Control Color Index	CCI_BACKGROUND
Presentation Parameter	PP_MENUBACKGROUNDCOLOR
System Default	SYSCLR_MENU
Remarks	None

- Text color

Control Color Index	CCI_FOREGROUND
Presentation Parameter	PP_MENUFOREGROUNDCOLOR
System Default	SYSCLR_MENUTEXT
Remarks	Pure RGB color.

- Disabled background color

Control Color Index	CCI_DISABLEDBACKGROUND
Presentation Parameter	PP_MENUDISABLEDBGNDCOLOR
System Default	SYSCLR_MENU
Remarks	None

- Disabled text color

Control Color Index	CCI_DISABLEDFOREGROUND
Presentation Parameter	PP_MENUDISABLEDFGNDCOLOR
System Default	SYSCLR_MENUDISABLEDTEXT
Remarks	Pure RGB color.

- Highlight background color

Control Color Index	CCI_HIGHLIGHTBACKGROUND
Presentation Parameter	PP_MENUHILITEBGNDCOLOR
System Default	SYSCLR_MENUHILITEBGND
Remarks	None

- Highlight text color

Control Color Index	CCI_HIGHLIGHTFOREGROUND
Presentation Parameter	PP_MENUHILITEFGNDCOLOR
System Default	SYSCLR_MENUHILITE
Remarks	Pure RGB color.

- Light border color

Control Color Index	CCI_BORDERLIGHT
Presentation Parameter	PP_BORDERLIGHTCOLOR
System Default	RGB_WHITE
Remarks	None

- Dark border color

Control Color Index	CCI_BORDERDARK
Presentation Parameter	PP_BORDERDARKCOLOR
System Default	RGB_DARKGRAY
Remarks	RGB_DARKGRAY not defined in OS/2 header files. Its value is 0x00808080.

- Light border 2 color

Control Color Index	CCI_BORDER2LIGHT
Presentation Parameter	PP_BORDER2LIGHTCOLOR
System Default	RGB_DARKGRAY
Remarks	RGB_DARKGRAY not defined in OS/2 header files. Its value is 0x00808080.

- Dark border 2 color

Control Color Index	CCI_BORDER2DARK
Presentation Parameter	PP_BORDER2DARKCOLOR
System Default	RGB_BLACK
Remarks	None

- Arrow color

Control Color Index	CCI_ARROW
Presentation Parameter	PP_ARROWCOLOR
System Default	RGB_DARKGRAY
Remarks	RGB_DARKGRAY not defined in OS/2 header files. Its value is 0x00808080.

- Light arrow border color

Control Color Index	CCI_ARROWBORDERLIGHT
Presentation Parameter	PP_ARROWBORDERLIGHTCOLOR
System Default	RGB_WHITE
Remarks	None

- Dark arrow border color

Control Color Index	CCI_ARROWBORDERDARK
Presentation Parameter	PP_ARROWBORDERDARKCOLOR

System Default	RGB_BLACK
Remarks	None

- Light check mark color

Control Color Index	CCI_CHECKLIGHT
Presentation Parameter	PP_CHECKLIGHTCOLOR
System Default	RGB_WHITE
Remarks	None
- Middle check mark color

Control Color Index	CCI_CHECKMIDDLE
Presentation Parameter	PP_CHECKMIDDLECOLOR
System Default	RGB_DARKGRAY
Remarks	RGB_DARKGRAY not defined in OS/2 header files. Its value is 0x00808080.
- Dark check mark color

Control Color Index	CCI_CHECKDARK
Presentation Parameter	PP_CHECKDARKCOLOR
System Default	RGB_BLACK
Remarks	None

Frame : CCT_FRAME

Color Usage Description	
Background	
Dialog background	
Inactive size border	
Inactive dialog border	
Active size border	
Active dialog border	
Thin border	
Light border	
Dark border	
Light border 2	
Dark border 2	
Icon text	
Icon text background	
Icon text background (on desktop)	
Icon text highlight	

Icon text background highlight

- **Background color**

Control Color Index	CCI_BACKGROUND
Presentation Parameter	PP_BACKGROUNDCOLOR
System Default	SYSCLR_WINDOW
Remarks	None

- **Dialog background color**

Control Color Index	CCI_BACKGROUNDDDIALOG
Presentation Parameter	PP_BACKGROUNDCOLOR
System Default	SYSCLR_DIALOGBACKGROUND
Remarks	None

- **Inactive size border color**

Control Color Index	CCI_INACTIVEFOREGROUND
Presentation Parameter	PP_INACTIVECOLOR
System Default	SYSCLR_INACTIVEBORDER
Remarks	None

- **Inactive dialog border color**

Control Color Index	CCI_INACTIVEFOREGROUNDIALOG
Presentation Parameter	PP_INACTIVECOLOR
System Default	SYSCLR_INACTIVEBORDER
Remarks	Not inherited.

- **Active size border**

Control Color Index	CCI_ACTIVEFOREGROUND
Presentation Parameter	PP_ACTIVECOLOR
System Default	SYSCLR_ACTIVEBORDER
Remarks	None

- **Active dialog border**

Control Color Index	CCI_ACTIVEFOREGROUNDIALOG
Presentation Parameter	PP_ACTIVECOLOR
System Default	SYSCLR_ACTIVEBORDER

Remarks	Not inherited.
• Thin border color	
Control Color Index	CCI_BORDER
Presentation Parameter	PP_BORDERCOLOR
System Default	SYSCLR_WINDOWFRAME
Remarks	None
• Light border color	
Control Color Index	CCI_BORDERLIGHT
Presentation Parameter	PP_BORDERLIGHTCOLOR
System Default	RGB_WHITE
Remarks	None
• Dark border color	
Control Color Index	CCI_BORDERDARK
Presentation Parameter	PP_BORDERDARKCOLOR
System Default	RGB_DARKGRAY
Remarks	RGB_DARKGRAY not defined in OS/2 header files. Its value is 0x00808080.
• Light border 2 color	
Control Color Index	CCI_BORDER2LIGHT
Presentation Parameter	PP_BORDER2LIGHTCOLOR
System Default	RGB_DARKGRAY
Remarks	RGB_DARKGRAY not defined in OS/2 header files. Its value is 0x00808080.
• Dark border 2 color	
Control Color Index	CCI_BORDER2DARK
Presentation Parameter	PP_BORDER2DARKCOLOR
System Default	RGB_BLACK
Remarks	None
• Icon text color	

Control Color Index	CCI_ICONFOREGROUND
Presentation Parameter	PP_INACTIVETEXTFGNDCOLOR
System Default	SYSCLR_ICONTEXT
Remarks	Not inherited

- Icon text background color

Control Color Index	CCI_ICONBACKGROUND
Presentation Parameter	PP_INACTIVETEXTBGNDCOLOR
System Default	SYSCLR_APPWORKSPACE
Remarks	Not inherited.

- Icon text background color (on desktop)

Control Color Index	CCI_ICONBACKGROUNDDESKTOP
Presentation Parameter	PP_INACTIVETEXTBGNDCOLOR
System Default	SYSCLR_BACKGROUND
Remarks	Not inherited.

- Icon text highlight color

Control Color Index	CCI_ICONHILITEFOREGROUND
Presentation Parameter	PP_HILITEFOREGROUNDCOLOR
System Default	SYSCLR_HILITEFOREGROUND
Remarks	Not inherited.

- Icon text background highlight color

Control Color Index	CCI_ICONHILITEBACKGROUND
Presentation Parameter	PP_HILITEBACKGROUNDCOLOR
System Default	SYSCLR_HILITEBACKGROUND
Remarks	Not inherited.

Scroll Bar : CCT_SCROLLBAR

- Color Usage Description

Background

Disabled background
Border
Light border
Dark border
Light border 2
Dark border 2
Button background
Button light border
Button dark border
Arrow

- Background color

Control Color Index	CCI_BACKGROUND
Presentation Parameter	PP_BACKGROUNDCOLOR
System Default	SYSCLR_SCROLLBAR
Remarks	Not inherited.

- Disabled background color

Control Color Index	CCI_DISABLEDBACKGROUND
Presentation Parameter	PP_BACKGROUNDCOLOR
System Default	SYSCLR_FIELDBACKGROUND
Remarks	Not inherited.

- Border color

Control Color Index	CCI_BORDER
Presentation Parameter	PP_BORDERCOLOR
System Default	SYSCLR_WINDOWFRAME
Remarks	None

- Light border color

Control Color Index	CCI_BORDERLIGHT
Presentation Parameter	PP_BORDERLIGHTCOLOR
System Default	RGB_WHITE
Remarks	None

- Dark border color

Control Color Index	CCI_BORDERDARK
Presentation Parameter	PP_BORDERDARKCOLOR
System Default	RGB_DARKGRAY
Remarks	RGB_DARKGRAY not defined in OS/2 header files. Its value is 0x00808080.

- **Light border 2 color**

Control Color Index	CCI_BORDER2LIGHT
Presentation Parameter	PP_BORDER2LIGHTCOLOR
System Default	RGB_WHITE
Remarks	None

- **Dark border 2 color**

Control Color Index	CCI_BORDER2DARK
Presentation Parameter	PP_BORDER2DARKCOLOR
System Default	RGB_BLACK
Remarks	None

- **Button background color**

Control Color Index	CCI_BUTTONBACKGROUND
Presentation Parameter	PP_HILITEFOREGROUNDCOLOR
System Default	SYSCLR_BUTTONMIDDLE
Remarks	Not inherited.

- **Button light border color**

Control Color Index	CCI_BUTTONBORDERLIGHT
Presentation Parameter	PP_BUTTONBORDERLIGHTCOLOR
System Default	SYSCLR_BUTTONLIGHT
Remarks	Not inherited.

- **Button dark border color**

Control Color Index	CCI_BUTTONBORDERDARK
Presentation Parameter	PP_BUTTONBORDERDARKCOLOR
System Default	SYSCLR_BUTTONDARK
Remarks	Not inherited.

- **Arrow color**

Control Color Index	CCI_ARROW
Presentation Parameter	PP_ARROWCOLOR
System Default	RGB_BLACK

Remarks	None
---------	------

Spin Button : CCT_SPINBUTTON

- Color Usage Description

Interior border
Light border
Dark border
Light border 2
Dark border 2
Button background
Button light border
Button dark border
Arrow

- Interior border color

Control Color Index	CCI_BORDER
Presentation Parameter	PP_BORDERCOLOR
System Default	RGB_BLACK
Remarks	None

- Light border color

Control Color Index	CCI_BORDERLIGHT
Presentation Parameter	PP_BORDERLIGHTCOLOR
System Default	RGB_WHITE
Remarks	None

- Dark border color

Control Color Index	CCI_BORDERDARK
Presentation Parameter	PP_BORDERDARKCOLOR
System Default	RGB_DARKGRAY
Remarks	RGB_DARKGRAY not defined in OS/2 header files. Its value is 0x00808080.

- Light border 2 color

Control Color Index	CCI_BORDER2LIGHT
Presentation Parameter	PP_BORDER2LIGHTCOLOR
System Default	RGB_LIGHTGRAY

Remarks	RGB_LIGHTGRAY not defined in OS/2 header files. Its value is 0x00cccccc.
• Dark border 2 color	
Control Color Index	CCI_BORDER2DARK
Presentation Parameter	PP_BORDER2DARKCOLOR
System Default	RGB_BLACK
Remarks	None
• Button background color	
Control Color Index	CCI_BUTTONBACKGROUND
Presentation Parameter	PP_BUTTONBACKGROUNDCOLOR
System Default	SYSCLR_BUTTONMIDDLE
Remarks	None
• Button light border color	
Control Color Index	CCI_BUTTONBORDERLIGHT
Presentation Parameter	PP_BUTTONBORDERLIGHTCOLOR
System Default	SYSCLR_BUTTONLIGHT
Remarks	None
• Button dark border color	
Control Color Index	CCI_BUTTONBORDERDARK
Presentation Parameter	PP_BUTTONBORDERDARKCOLOR
System Default	SYSCLR_BUTTONDARK
Remarks	None
• Arrow color	
Control Color Index	CCI_ARROW
Presentation Parameter	PP_ARROWCOLOR
System Default	RGB_BLACK
Remarks	None

Slider : CCT_SLIDER

- Color Usage Description

- Background
- Text / detent
- Shaft ribbon color
- Focus highlight
- Light border
- Dark border
- Light tick mark
- Dark tick mark
- Button background
- Button light border
- Button dark border
- Arrow

- Text / detent

Control Color Index	CCI_FOREGROUND
Presentation Parameter	PP_FOREGROUNDCOLOR
System Default	SYSCLR_WINDOWTEXT
Remarks	None

- Shaft ribbon color

Control Color Index	CCI_BACKGROUND
Presentation Parameter	PP_HILITEBACKGROUNDCOLOR
System Default	0x000000AA
Remarks	None

- Focus highlight color

Control Color Index	CCI_BORDER
Presentation Parameter	PP_BORDERCOLOR
System Default	RGB_BLACK
Remarks	None

- Light border color

Control Color Index	CCI_BORDERLIGHT
Presentation Parameter	PP_BORDERLIGHTCOLOR
System Default	RGB_WHITE
Remarks	None

- Dark border color

Control Color Index	CCI_BORDERDARK
Presentation Parameter	PP_BORDERDARKCOLOR
System Default	RGB_DARKGRAY
Remarks	RGB_DARKGRAY not defined in OS/2 header files. Its value is 0x00808080.

- Light tick mark color

Control Color Index	CCI_BORDER2LIGHT
Presentation Parameter	PP_BORDER2LIGHTCOLOR
System Default	RGB_WHITE
Remarks	None

- Dark tick mark color

Control Color Index	CCI_BORDER2DARK
Presentation Parameter	PP_BORDER2DARKCOLOR
System Default	RGB_DARKGRAY
Remarks	RGB_DARKGRAY not defined in OS/2 header files. Its value is 0x00808080.

- Button background color

Control Color Index	CCI_BUTTONBACKGROUND
Presentation Parameter	PP_BUTTONBACKGROUNDCOLOR
System Default	SYSCLR_BUTTONMIDDLE
Remarks	None

- Button light border color

Control Color Index	CCI_BUTTONBORDERLIGHT
Presentation Parameter	PP_BUTTONBORDERLIGHTCOLOR
System Default	SYSCLR_BUTTONLIGHT
Remarks	None

- Button dark border color

Control Color Index	CCI_BUTTONBORDERDARK
Presentation Parameter	PP_BUTTONBORDERDARKCOLOR
System Default	SYSCLR_BUTTONDARK
Remarks	None

- Arrow color

Control Color Index	CCI_ARROW
Presentation Parameter	PP_ARROWCOLOR
System Default	RGB_BLACK
Remarks	None

Circular Slider : CCT_CIRCULARSLIDER

- Color Usage Description

Background
 Text
 Border
 Light border
 Dark border
 Disabled border
 Disabled light border
 Disabled dark border
 Button background
 Button light border
 Button dark border
 Arrow/tick marks
 Disabled arrow/tick marks

- Background color

Control Color Index	CCI_BACKGROUND
Presentation Parameter	PP_BACKGROUNDCOLOR
System Default	SYSCLR_FIELDBACKGROUND
Remarks	None

- Text color

Control Color Index	CCI_FOREGROUND
Presentation Parameter	PP_FOREGROUNDCOLOR
System Default	SYSCLR_WINDOWTEXT
Remarks	None

- Border color

Control Color Index	CCI_BORDER
Presentation Parameter	PP_BORDERCOLOR

	System Default	RGB_BLACK
	Remarks	None
•	Light border color	
	Control Color Index	CCI_BORDERLIGHT
	Presentation Parameter	PP_BORDERLIGHTCOLOR
	System Default	RGB_WHITE
	Remarks	None
•	Dark border color	
	Control Color Index	CCI_BORDERDARK
	Presentation Parameter	PP_BORDERDARKCOLOR
	System Default	RGB_DARKGRAY
	Remarks	RGB_DARKGRAY not defined in OS/2 header files. Its value is 0x00808080.
•	Disabled border color	
	Control Color Index	CCI_BORDER2
	Presentation Parameter	PP_BORDER2COLOR
	System Default	SYSCLR_BUTTONMIDDLE
	Remarks	None
•	Disabled light border color	
	Control Color Index	CCI_BORDER2LIGHT
	Presentation Parameter	PP_BORDER2LIGHTCOLOR
	System Default	RGB_WHITE
	Remarks	None
•	Disabled dark border color	
	Control Color Index	CCI_BORDER2DARK
	Presentation Parameter	PP_BORDER2DARKCOLOR
	System Default	SYSCLR_BUTTONMIDDLE
	Remarks	None
•	Button background color	

Control Color Index	CCI_BUTTONBACKGROUND
Presentation Parameter	PP_BUTTONBACKGROUNDCOLOR
System Default	SYSCLR_BUTTONMIDDLE
Remarks	None

- Button light border color

Control Color Index	CCI_BUTTONBORDERLIGHT
Presentation Parameter	PP_BUTTONBORDERLIGHTCOLOR
System Default	SYSCLR_BUTTONLIGHT
Remarks	None

- Button dark border color

Control Color Index	CCI_BUTTONBORDERDARK
Presentation Parameter	PP_BUTTONBORDERDARKCOLOR
System Default	SYSCLR_BUTTONDARK
Remarks	None

- Arrow/tick marks color

Control Color Index	CCI_ARROW
Presentation Parameter	PP_ARROWCOLOR
System Default	RGB_BLACK
Remarks	None

- Disabled arrow/tick marks color

Control Color Index	CCI_DISABLEDARROW
Presentation Parameter	PP_ARROWDISABLEDCOLOR
System Default	SYSCLR_BUTTONDARK
Remarks	None

MLE : CCT_MLE

- Color Usage Description
 - Background

Text
Read-only text
Highlight background
Highlight text
Light border
Dark border
Light border 2
Dark border 2
Corner between scroll bars

- Background color

Control Color Index	CCI_BACKGROUND
Presentation Parameter	PP_BACKGROUNDCOLOR
System Default	SYSCLR_ENTRYFIELD
Remarks	Not inherited.

- Text color

Control Color Index	CCI_FOREGROUND
Presentation Parameter	PP_FOREGROUNDCOLOR
System Default	SYSCLR_WINDOWTEXT
Remarks	None

- Read-only text color

Control Color Index	CCI_FOREGROUNDREADONLY
Presentation Parameter	PP_FOREGROUNDREADONLYCOLOR
System Default	SYSCLR_OUTPUTTEXT
Remarks	None

- Highlight background color

Control Color Index	CCI_HIGHLIGHTBACKGROUND
Presentation Parameter	PP_HILITEBACKGROUNDCOLOR
System Default	SYSCLR_HILITEBACKGROUND
Remarks	None

- Text color

Control Color Index	CCI_HIGHLIGHTFOREGROUND
Presentation Parameter	PP_HILITEFOREGROUNDCOLOR
System Default	SYSCLR_HILITEFOREGROUND
Remarks	None

- Light border color

- | | |
|------------------------|---------------------|
| Control Color Index | CCI_BORDERLIGHT |
| Presentation Parameter | PP_BORDERLIGHTCOLOR |
| System Default | RGB_WHITE |
| Remarks | None |
- Dark border color

Control Color Index	CCI_BORDERDARK
Presentation Parameter	PP_BORDERDARKCOLOR
System Default	RGB_DARKGRAY
Remarks	RGB_DARKGRAY not defined in OS/2 header files. Its value is 0x00808080.
 - Light border 2 color

Control Color Index	CCI_BORDER2LIGHT
Presentation Parameter	PP_BORDER2LIGHTCOLOR
System Default	RGB_LIGHTGRAY
Remarks	RGB_LIGHTGRAY is not defined in OS/2 header files. Its value is 0x00CCCCC.
 - Dark border 2 color

Control Color Index	CCI_BORDER2DARK
Presentation Parameter	PP_BORDER2DARKCOLOR
System Default	RGB_BLACK
Remarks	None
 - Corner between scroll bars color

Control Color Index	CCI_FIELDBACKGROUND
Presentation Parameter	PP_FIELDBACKGROUNDCOLOR
System Default	SYSCLR_FIELDBACKGROUND
Remarks	None

Value Set : CCT_VALUESET

- Color Usage Description

Background
Text
Border
Default border

- Background color

Control Color Index	CCI_BACKGROUND
Presentation Parameter	PP_BACKGROUNDCOLOR
System Default	SYSCLR_FIELDBACKGROUND
Remarks	None

- Text color

Control Color Index	CCI_FOREGROUND
Presentation Parameter	PP_FOREGROUNDCOLOR
System Default	SYSCLR_WINDOWTEXT
Remarks	None

- Border color

Control Color Index	CCI_BORDER
Presentation Parameter	PP_BORDERCOLOR
System Default	SYSCLR_WINDOWFRAME
Remarks	None

- Default border color

Control Color Index	CCI_BORDERDEFAULT
Presentation Parameter	PP_HILITEBACKGROUNDCOLOR
System Default	SYSCLR_HILITEBACKGROUND
Remarks	None

Notebook : CCT_NOTEBOOK

- Color Usage Description

Background
Text

Status line background
Page background
Selection cursor
Outline
Light border
Dark border
New notebook border background
Light tab border
Dark tab border
Major tab background
Major tab text
Minor tab background
Minor tab text
Dog-ear dark shadow
Arrow

- Background color

Control Color Index	CCI_BACKGROUND
Presentation Parameter	PP_BACKGROUNDCOLOR
System Default	SYSCLR_FIELDBACKGROUND
Remarks	None

- Text color

Control Color Index	CCI_FOREGROUND
Presentation Parameter	PP_FOREGROUNDCOLOR
System Default	SYSCLR_WINDOWTEXT
Remarks	None

- Status line background color

Control Color Index	CCI_FIELDBACKGROUND
Presentation Parameter	PP_BACKGROUNDCOLOR
System Default	SYSCLR_FIELDBACKGROUND
Remarks	None

- Page background color

Control Color Index	CCI_PAGEBACKGROUND
Presentation Parameter	PP_PAGEBACKGROUNDCOLOR
System Default	SYSCLR_PAGEBACKGROUND
Remarks	None

- Selection cursor color

Control Color Index	CCI_BORDERDEFAULT
Presentation Parameter	PP_HILITEBACKGROUNDCOLOR

	System Default	SYSCLR_BUTTONDEFAULT
	Remarks	None
•	Outline color	
	Control Color Index	CCI_BORDER
	Presentation Parameter	PP_BORDERCOLOR
	System Default	SYSCLR_WINDOWFRAME
	Remarks	None
•	Light border color	
	Control Color Index	CCI_BORDERLIGHT
	Presentation Parameter	PP_BORDERLIGHTCOLOR
	System Default	RGB_WHITE
	Remarks	None
•	Dark border color	
	Control Color Index	CCI_BORDERDARK
	Presentation Parameter	PP_BORDERDARKCOLOR
	System Default	RGB_DARKGRAY
	Remarks	RGB_DARKGRAY not defined in OS/2 header files. Its value is 0x00808080.
•	New notebook border background	
	Control Color Index	CCI_BORDER2
	Presentation Parameter	PP_BORDER2COLOR
	System Default	SYSCLR_FIELDBACKGROUND
	Remarks	None
•	Light tab border color	
	Control Color Index	CCI_BORDER2LIGHT
	Presentation Parameter	PP_BORDER2LIGHTCOLOR
	System Default	RGB_WHITE
	Remarks	None
•	Dark tab border color	

	Control Color Index	CCI_BORDER2DARK
	Presentation Parameter	PP_BORDER2DARKCOLOR
	System Default	RGB_DARKGRAY
	Remarks	RGB_DARKGRAY not defined in OS/2 header files. Its value is 0x00808080.
•	Major tab background color	
	Control Color Index	CCI_MAJORTABBACKGROUND
	Presentation Parameter	PP_MAJORTABBACKGROUNDCOLOR
	System Default	SYSCLR_PAGEBACKGROUND
	Remarks	None
•	Major tab text color	
	Control Color Index	CCI_MAJORTABFOREGROUND
	Presentation Parameter	PP_MAJORTABFOREGROUNDCOLOR
	System Default	SYSCLR_WINDOWTEXT
	Remarks	None
•	Minor tab background color	
	Control Color Index	CCI_MINORTABBACKGROUND
	Presentation Parameter	PP_MINORTABBACKGROUNDCOLOR
	System Default	SYSCLR_PAGEBACKGROUND
	Remarks	None
•	Minor tab text color	
	Control Color Index	CCI_MINORTABFOREGROUND
	Presentation Parameter	PP_MINORTABFOREGROUNDCOLOR
	System Default	SYSCLR_WINDOWTEXT
	Remarks	None
•	Dog-ear dark shadow color	
	Control Color Index	CCI_CHECKDARK
	Presentation Parameter	PP_CHECKDARKCOLOR
	System Default	RGB_BLACK
	Remarks	None

- Arrow color

Control Color Index	CCI_ARROW
Presentation Parameter	PP_ARROWCOLOR
System Default	RGB_BLUE
Remarks	None

Container : CCT_CONTAINER

- Color Usage Description

Background
 Text
 Icon text
 Icon text background
 Background highlight
 Text highlight
 Border
 Edit background
 Edit text
 Tree view light button border
 Tree view dark button border
 Tree view outer button border
 Tree view button background
 Record emphasis
 Edge window

- Background color

Control Color Index	CCI_BACKGROUND
Presentation Parameter	PP_BACKGROUNDCOLOR
System Default	SYSCLR_WINDOW
Remarks	None

- Text color

Control Color Index	CCI_FOREGROUND
Presentation Parameter	PP_FOREGROUNDCOLOR
System Default	SYSCLR_WINDOWTEXT
Remarks	None

- Icon text color

Control Color Index	CCI_ICONFOREGROUND
Presentation Parameter	PP_FOREGROUNDCOLOR

	System Default	SYSCLR_ICONTEXT
	Remarks	Pure RGB color and not inherited.
•	Icon text background color	
	Control Color Index	CCI_ICONBACKGROUND
	Presentation Parameter	PP_ICONTEXTBACKGROUNDCOLOR
	System Default	CLR_ERROR (transparent)
	Remarks	None
•	Background highlight color	
	Control Color Index	CCI_HIGHLIGHTBACKGROUND
	Presentation Parameter	PP_HILITEBACKGROUNDCOLOR
	System Default	SYSCLR_HILITEBACKGROUND
	Remarks	None
•	Text highlight color	
	Control Color Index	CCI_HIGHLIGHTFOREGROUND
	Presentation Parameter	PP_HILITEFOREGROUNDCOLOR
	System Default	SYSCLR_HILITEFOREGROUND
	Remarks	None
•	Edit background color	
	Control Color Index	CCI_PAGEBACKGROUND
	Presentation Parameter	PP_PAGEBACKGROUNDCOLOR
	System Default	RGB_WHITE
	Remarks	None
•	Edit foreground color	
	Control Color Index	CCI_PAGEFOREGROUND
	Presentation Parameter	PP_PAGEFOREGROUNDCOLOR
	System Default	RGB_BLACK
	Remarks	None
•	Light border color	
	Tree view light button border	CCI_BUTTONBORDERLIGHT

	color	
	Presentation Parameter	PP_BUTTONBORDERLIGHTCOLOR
	System Default	SYSCLR_BUTTONLIGHT
	Remarks	None
•	Tree view dark button border color	
	Control Color Index	CCI_BUTTONBORDERDARK
	Presentation Parameter	PP_BUTTONBORDERDARKCOLOR
	System Default	SYSCLR_BUTTONDARK
	Remarks	None
•	Tree view outer button border color	
	Control Color Index	CCI_BORDERDEFAULT
	Presentation Parameter	PP_BORDERDEFAULTCOLOR
	System Default	RGB_BLACK
	Remarks	None
•	Tree view button background color	
	Control Color Index	CCI_BUTTONBACKGROUND
	Presentation Parameter	PP_BUTTONBACKGROUNDCOLOR
	System Default	SYSCLF_BUTTONMIDDLE
	Remarks	None
•	Record emphasis color	
	Control Color Index	CCI_BORDER2
	Presentation Parameter	PP_BORDER2COLOR
	System Default	RGB_BLACK
	Remarks	None
•	Edge window color	
	Control Color Index	CCI_FIELDBACKGROUND
	Presentation Parameter	PP_FIELDBACKGROUNDCOLOR
	System Default	SYSCLR_SCROLLBAR
	Remarks	None
•	Major tab text color	

Control Color Index	CCI_MAJORTABFOREGROUND
Presentation Parameter	PP_MAJORTABFOREGROUND
System Default	SYSCLR_WINDOWTEXT
Remarks	None

- Minor tab background color

Control Color Index	CCI_MINORTABBACKGROUND
Presentation Parameter	PP_MINORTABBACKGROUND
System Default	SYSCLR_PAGEBACKGROUND
Remarks	None

- Minor tab text color

Control Color Index	CCI_MINORTABFOREGROUND
Presentation Parameter	PP_MINORTABFOREGROUND
System Default	SYSCLR_WINDOWTEXT
Remarks	None

- Dog-ear dark shadow color

Control Color Index	CCI_CHECKDARK
Presentation Parameter	PP_CHECKDARKCOLOR
System Default	RGB_BLACK
Remarks	None

- Arrow color

Control Color Index	CCI_ARROW
Presentation Parameter	PP_ARROWCOLOR
System Default	RGB_BLUE
Remarks	None

Overriding the Default Colors Used by PM Controls

There is a way to override the default colors used by PM controls. WinSetControlColors can be used to set one or more *control colors*. Control colors can be set globally for all instances of the control class (CCF_GLOBAL), or for all instances of the control created on the

same thread (CCF_APPLICATION). For example, you can set the background color of all the push buttons in your application to brown without affecting the global default push button background color (normally light-gray). Setting a control color for an individual control window causes the appropriate presentation parameter to be set. Each control has a control color type, and a set of control color indexes are used for each type to provide access to the colors used in the control.

WinQueryControlColors allows an application to query one or more of the colors being used by a PM control. When your application uses this function in conjunction with WinSetControlColors, it can query and then set all of the colors used in a PM system control window at one time. This is more convenient than using presentation parameters, but it is not backwardly compatible with versions earlier than OS/2 Warp Version 4.0.

WM_CTLCOLORCHANGE is sent to a window when a control color has been changed (similar to WM_SYSCOLORCHANGE), and WM_QUERYCTLTYPE can be sent to a window to find out which, if any, control colors it responds to (a control returns a non-zero index value corresponding to a CCT_ constant).

```

/*****
 * Query the number of colors used by a push button.
 *****/

cCount = WinQueryControlColors(
    HWND_DESKTOP,          /* Desktop window handle */
    CCT_PUSHBUTTON,        /* Select push button */
    CCF_COUNTCOLORS,       /* Count number of colors */
    0, 0);

pactlColor = (PCTL_COLOR)malloc(sizeof(CTL_COLOR) * cCount);

/*****
 * Query all the colors used by push buttons in application.
 *****/

WinQueryControlColors(HWND_DESKTOP,          /* Desktop window handle */
    CCT_PUSHBUTTON,        /* Select push button */
    CCF_ALLCOLORS |        /* Return all colors ... */
    CCF_APPLICATION,       /* ... for application */
    cCount,                /* Size of array */
    pactlColor);           /* Buffer for color data */

/*****
 * Give the global push button borders a red color.
 *****/

for (i = 0; i < cCount; i++)
{
    switch (pactlColor[i].clrIndex)
    {
        case CCI_BORDERLIGHT:

            pactlColor[i].clrValue = 0x00FFC0C0; /* Light red */
            break;

        case CCI_BORDERDARK:

            pactlColor[i].clrValue = 0x00C00000; /* Dark red */
            break;

        default:

            pactlColor[i].clrValue = CCV_DEFAULT; /* Default color */
            break;
    }
}

/*****
 * Set the new border colors for all push buttons in application.
 *****/

WinSetControlColors(HWND_DESKTOP,          /* Desktop window handle */
    CCT_PUSHBUTTON,        /* Select push button */
    CCF_APPLICATION,       /* Application colors */
    cCount,                /* Number of colors */
    pactlColor);           /* Buffer for color data */

```

Resource Files

Resource files enable you to specify the resource information used in creating an application's window. Some examples of resources that can be defined in resource files are:

- Menus
- Accelerator tables
- Dialog and window templates
- Icons
- Fonts
- Bit maps
- Strings

To add resource information to an application, use a text editor to create a *resource* script file, and then compile it using the *Resource Compiler*, RC.EXE. The advantage of using resource files is that resource information can be maintained and updated separately in the resource script file and then linked to your application program's .EXE file. This greatly simplifies customizing an application because you can modify resource information without having to recompile the entire application.

This chapter describes the use of resource files in Presentation Manager (PM) programming.

About Resource Files

A resource script file is a text file that contains one or more resource statements that define the type, identifier, and data for each resource. Because some resources might contain binary data that cannot be created using a text editor, there are resource statements that let you specify additional files to include when compiling the resource script file. For example, you can use the *Dialog Box Editor* to design dialog boxes, the *Font Editor* to edit font files, and the *Icon Editor* to create customized icons, pointers, and bit maps. The definitions for these resources can be included with other resource definitions in the resource file.

Resource Statements

This section provides overview information on resource statements and directives. Resource statements consist of one or more keywords, numbers, character strings, constants, or file names. You combine these to define the resource type, identifier, and data. Directives are special types of resource statements that perform functions such as including header files, defining constants, and conditionally compiling portions of the file. Resource statements have three basic forms:

- Single-line statements
 - Multiple-line statements
 - Directives
-

Single-line Statements

Single-line statements consist of a keyword identifying the resource type, a constant or number specifying the resource identifier, and a file name specifying the file containing the resource data. For example, this ICON statement defines an icon resource:

```
ICON 1 myicon.ico
```

The icon resource has the icon identifier 1, and the file MYICON.ICO contains the icon data.

Multiple-line Statements

Multiple-line statements consist of a keyword identifying the resource type, a constant or number specifying the resource identifier, and, between the BEGIN and END keywords, additional resource statements that define the resource data. For example, this MENU statement defines a menu resource:

```
MENU 1
BEGIN
    MENUITEM "Alpha", 101
    MENUITEM "Beta", 102
END
```

The menu identifier is 1. The menu contains two MENUITEM statements that define the contents of the menu.

In multiple-line statements such as DLGTEMPLATE and WINDOWTEMPLATE, any level of nested statements is allowed. For example, the DLGTEMPLATE and WINDOWTEMPLATE statements typically contain a single DIALOG or FRAME statement. These statements can contain any number of WINDOW and CONTROL statements; the WINDOW and CONTROL statements can contain additional WINDOW and CONTROL statements, and so forth. The nested statements let you define controls and other child windows for the dialog boxes and windows.

If a nested statement creates a child window or control, the parent and owner of the new window is the window created by the containing statement. (FRAME statements occasionally create frame controls whose parent and owner windows are not the same.)

Directives

Directives consist of the reserved character # in the first column of a line, followed by the directive keyword and any additional numbers, character strings, or file names.

Some examples of directives are:

- #define
- #if
- #ifdef
- #include

Descriptions of the individual directives follow the resource file statement descriptions.

Resource File Statement Descriptions

This section provides the syntax, description, and an example of each of the resource file statements.

The following table summarizes, at a general level, the most commonly used parameters on the statements.

Parameter	Description
id	Control identifier.
x	X coordinate of the lower-left corner of the control.
y	Y coordinate of the lower-left corner of the control.
height	Height of the control (in 1/8 character units).
width	Width of the control.
style	Predefined bit representation of a style or

combination of styles.

load option	Definition of when the system should load the resource into memory (for example, PRELOAD or LOADONCALL).
mem option	Definition of how the system manages the resource when in memory (for example, FIXED, MOVABLE, or DISCARDABLE).
text	Text associated with a control.
class	Predefined class for a particular control.

ACCELTABLE Statement

The ACCELTABLE statement creates a table of accelerators for an application.

Syntax

```
ACCELTABLE acceltable-id [mem-option][load-option]
BEGIN
key-value, command[, accelerator-options] ...
.
.
.
END
```

Description

An accelerator is a keystroke that gives the user a quick way to choose a command from a menu or carry out some other task. An accelerator table can be loaded when needed from the executable file by using the WinLoadAccelTable function.

Example

This example creates an accelerator table whose accelerator-table identifier is 1. The table contains two accelerators: Ctrl+S and Ctrl+G. These accelerators generate WM_COMMAND messages with values of 101 and 102, respectively, when the user presses the corresponding keys.

```
ACCELTABLE 1
BEGIN
"S", 101, CONTROL
"G", 102, CONTROL
END
```

ASSOCTABLE Statement

The ASSOCTABLE statement defines a file-association table for an application.

Syntax

```
ASSOCTABLE assoctable-id [load-option][mem-option]
BEGIN
association-name, file-match-string[, extended-attribute-flag]
    [, icon-filename]
    .
    .
    .
END
```

Description

This table associates the data files that an application creates with the executable file of the application. When the user selects one of these data files, the associated application begins executing.

A file-association table can also associate icons with the data files that an application creates. The icons are used to identify the data files graphically. Because a file-association table associates icons by file type, all data files having the same file type have the same icon.

You can provide any number of ASSOCTABLE statements in a resource script file, but each statement must specify a unique assoctable-id value. The file-association tables are written not only to the resources within your executable file, but also to the .ASSOC extended attribute. However, only the last file-association table specified in the resource script file is actually written to the extended attribute.

AUTOCHECKBOX Statement

The AUTOCHECKBOX statement creates an automatic-check-box control.

Syntax

```
AUTOCHECKBOX text, id, x, y, width [, style]
```

Description

The control is a small rectangle (check box) that contains an X when the user selects it. The specified text is displayed to the right of the check box. An X appears in the square when the user first selects the control and disappears the next time the user selects it. The `AUTOCHECKBOX` statement, which can only be used in a `DIALOG` or `WINDOW` statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is `WC_BUTTON`. If the style is not specified, the default style is `BS_AUTOCHECKBOX` and `WS_TABSTOP`.

Example

This example creates an automatic-check-box control that is labeled "Italic."

```
AUTOCHECKBOX "Italic", 101, 10, 10, 100, 100
```

AUTORADIOBUTTON Statement

The `AUTORADIOBUTTON` statement creates an automatic-radio-button control.

Syntax

```
AUTORADIOBUTTON text, id, x, y, width, height [, style]
```

Description

This control is a small circle with the given text displayed to its right. The control highlights the circle and sends a message to its parent window when the user selects the button. The control also removes the selection from any other automatic-radio-button controls in the same group. When the user selects the button again, the control removes the highlight before sending a message. The `AUTORADIOBUTTON` statement, which you can use only in a `DIALOG` or `WINDOW` statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is `WC_BUTTON`. If you do not specify a style, the default style is `BS_AUTORADIOBUTTON`.

Example

This example creates an automatic-radio-button control that is labeled "Italic."

```
AUTORADIOBUTTON "Italic", 101, 10, 10, 24, 50
```

BITMAP Statement

The BITMAP statement defines a bit-map resource for an application.

Syntax

```
BITMAP bitmap-id [load-option] [mem-option] filename
```

Description

A bit-map resource, typically created using the Icon Editor, is a custom bit map that an application uses in its display or as an item in a menu.

The BITMAP statement copies the bit-map resource from the file specified in the *filename* field and adds it to the application's other resources. A bit-map resource can be loaded from the executable file when needed by using the GpiLoadBitmap function.

You can provide any number of BITMAP statements in a resource script file, but each statement must specify a unique bitmap-id value.

Example

This example defines a bit map whose bit-map identifier is 12. The bit-map resource is copied from the file CUSTOM.BMP.

```
BITMAP 12 custom.bmp
```

CHECKBOX Statement

The CHECKBOX statement creates a check-box control.

Syntax

```
CHECKBOX text, id, x, y, width, height [, style]
```

Description

The control is a small rectangle (check box) that has the specified text displayed to the right. The control highlights the rectangle and sends a message to its parent window when the user selects the control. The CHECKBOX statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_BUTTON. If you do not specify a style, the default style is BS_CHECKBOX and WS_TABSTOP.

Example

This example creates a check-box control that is labeled "Italic."

```
CHECKBOX "Italic", 101, 10, 10, 100, 100
```

CODEPAGE Statement

The CODEPAGE statement sets the code page for all subsequent resources.

Syntax

```
CODEPAGE codepage-id
```

Description

The code page specifies the character set used for characters in the resource.

If the CODEPAGE statement is not given in a resource script file, the resource compiler uses the code page set up for the individual system. If more than one CODEPAGE statement is given in the file, each CODEPAGE statement applies to the resource statements between it and the next CODEPAGE statement.

Example

In this example, the code page for the character-string resources is set to Portuguese (860).

```
CODEPAGE 860  
  
STRINGTABLE
```

```
BEGIN
  1 "Filename not found"
  2 "Cannot open file for reading"
END
```

COMBOBOX Statement

The COMBOBOX statement creates a combination-box control.

Syntax

```
COMBOBOX text, id, x, y, width, height [, style]
```

Description

This control combines a list-box control with an entry-field control. It allows the user to place the selected item from a list box into an entry field.

The COMBOBOX statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_COMBOBOX. If you do not specify a style, the default style is CBS_SIMPLE, WS_GROUP, WS_TABSTOP, and WS_VISIBLE.

Example

This example creates a combination-box control.

```
COMBOBOX "", 101, 10, 10, 24, 50
```

CONTAINER Statement

The CONTAINER statement creates a container control within a dialog window.

Syntax

CONTAINER id, x, y, width, height [, style]

Description

The container control is a visual component that holds objects.

The CONTAINER statement defines the identifier, position, dimensions, and attributes of a container control. The predefined class for this control is WC_CONTAINER. If you do not specify a style, the default style is WS_TABSTOP, WS_VISIBLE, and CCS_SINGLESEL. A CONTAINER statement is only used in a DIALOG or WINDOW statement.

Example

This example creates a container control at position (30,30) within the dialog window. The container has a width of 70 character units and a height of 25 character units. Its resource ID is 301. The default style CCS_SINGLESEL has been overridden by the style specification CCS_MULTIPLESEL. The default styles WS_TABSTOP and WS_GROUP are both in effect, though only the latter is specified.

```
#define IDC_CONTAINER      301
#define IDD_CONTAINERDLG  504
DIALOG "Container", IDD_CONTAINERDLG, 23, 6, 120, 280, FS_NOBYTEALIGN |
    WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
BEGIN
    CONTAINER IDC_CONTAINER, 30, 30, 70, 200, CCS_MULTIPLESEL |
        WS_GROUP
END
```

CONTROL Statement

The CONTROL statement defines a control as belonging to the specified class.

Syntax

```
CONTROL text, id, x, y, width, height, class [, style]
[ data-definitions ] ...
[ BEGIN
control-definition
.
.
.
END ]
```

Description

The statement defines the position and dimensions of the control within the parent window, as well as the control style. The CONTROL statement is most often used in a DIALOG or WINDOW statement.

Typically, several CONTROL statements are used in each DIALOG statement, and each CONTROL statement must have a unique *id* value. The optional BEGIN and END statements enclose any CONTROL statements that may be given with the control. CONTROL statements given in this manner represent child windows belonging to the control created by the CONTROL statement.

The CONTROL statement can actually contain any combination of CONTROL, DIALOG, and WINDOW statements, but it usually does not contain such statements.

Example

This example creates a push-button control with the WS_TABSTOP and WS_VISIBLE styles.

```
CONTROL "OK", 101, 10, 10, 20, 50, WC_BUTTON, BS_PUSHBUTTON |
                                     WS_TABSTOP             |
                                     WS_VISIBLE               |
```

CTEXT Statement

The CTEXT statement creates a centered-text control.

Syntax

```
CTEXT text, id, x, y, width, height [, style]
```

Description

The control is a simple rectangle displaying the given text centered in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line.

The CTEXT statement defines the text, identifier, dimensions, and attributes of the control. The predefined class for this control is WC_STATIC. If you do not specify a style, the default style is SS_TEXT, DT_CENTER, and WS_GROUP.

Use the CTEXT statement only in a DIALOG or WINDOW statement.

Example

This example creates a centered-text control that is labeled "Filename."

```
CTEXT "Filename", 101, 10, 10, 100, 100
```

CTLDATA Statement

The CTLDATA statement defines control data for a custom dialog box, window, or control.

Syntax

```
CTLDATA word-value [, word-value] ...  
  
CTLDATA string  
  
CTLDATA MENU  
BEGIN  
menuitem-definition  
.  
.  
.  
END
```

Description

The statement has three basic forms to permit specifying a menu or specifying data in words or characters. The data can be in any format, because only your window procedure will use it. The window procedure of the dialog box, window, or control receives this data when the item is created. It is up to the window procedure to process the data.

CTLDATA is often used to supply data that controls the subsequent operation of the custom window. For example, the CTLDATA statement may contain extended style bits - that is, style bits designed specifically for your customized window.

You should reserve the CTLDATA statement for window classes that you create yourself.

Example

This example creates a menu for the window created with the WINDOW statement.

```
WINDOWTEMPLATE 1  
BEGIN  
    WINDOW "Sample", 1, 0, 0, 100, 100, "MYCLASS", 0, FCF_STANDARD  
    CTLDATA MENU  
    BEGIN  
        MENUITEM "Exit", 101  
    END  
END
```

DEFAULTICON Statement

This statement installs the named icon file definition under the ICON Extended Attribute of the program file.

Syntax

```
DEFAULTICON filename
```

Description

An icon with an icon-id of 1 is the default icon by default, unless you supply a different icon.

Example

```
DEFAULTICON filename.ico
```

DEFPUSHBUTTON Statement

The DEFPUSHBUTTON statement creates a default push-button control.

Syntax

```
DEFPUSHBUTTON text, id, x, y, width, height [, style]
```

Description

The control is a round-cornered rectangle containing the given text. The rectangle has a bold outline to represent that it is the default response for the user. The control sends a message to its parent window when the user chooses the control. The DEFPUSHBUTTON

statement defines the text, identifier, dimensions, and attributes of the control. The predefined class for this control is WC_BUTTON. If you do not specify a style, the default style is BS_PUSHBUTTON, BS_DEFAULT, and WS_TABSTOP.

Use the DEFPUSHBUTTON statement only in a DIALOG or WINDOW statement.

Example

This example creates a default push-button control that is labeled "Cancel."

```
DEFPUSHBUTTON "Cancel", 101, 10, 10, 24, 50
```

DIALOG Statement

The DIALOG statement defines a window that an application can use to create dialog boxes.

Syntax

```
DIALOG text, id, x, y, width, height [, [style] [,framectl]] [data-definitions]
BEGIN
control-definition
.
.
.
END
```

Description

The statement defines the position and dimensions of the dialog box on the screen, as well as the dialog-box style. The DIALOG statement is most often used in a DLGTEMPLATE statement.

Typically, you use only one DIALOG statement in each DLGTEMPLATE statement, and the DIALOG statement contains at least one control definition.

The exact meaning of the coordinates depends on the style defined by the *style* field. For dialog boxes with FS_SCREENALIGN style, the coordinates are relative to the origin of the display screen. For dialog boxes with the style FS_MOUSEALIGN, the coordinates are relative to the position of the mouse pointer at the time the dialog box is created. For all other dialog boxes, the coordinates are relative to the origin of the parent window.

The DIALOG statement can actually contain any combination of CONTROL, DIALOG, and WINDOW statements. Typically, a DIALOG statement contains one or more CONTROL statements.

Example

This example creates a dialog box that is labeled "Disk Error."

```
DLGTEMPLATE 1
BEGIN
    DIALOG  "Disk Error", 100, 10, 10, 300, 110
    BEGIN
        CTEXT "Select One:", 1, 10, 80, 280, 12
        RADIOBUTTON "Retry", 2, 75, 50, 60, 12
        RADIOBUTTON "Abort", 3, 75, 30, 60, 12
        RADIOBUTTON "Ignore", 4, 75, 10, 60, 12
    END
END
```

DLGINCLUDE Statement

The DLGINCLUDE statement adds the specified file name to the resource file.

Syntax

```
DLGINCLUDE id filename
```

Description

The DLGINCLUDE statement is typically used to let the application access the definitions file for the dialog box with the corresponding identifier. The file specified in the *filename* field must contain the define directives used by the dialog box.

You can provide any number of DLGINCLUDE statements in a resource script file, but each must have a unique identifier.

Example

This example includes the name of the definition file dlgdef.h. The dialog-box identifier is 5.

```
DLGINCLUDE 5 \\INCLUDE\\DLGDEF.H
```

DLGTEMPLATE Statement

The DLGTEMPLATE statement creates a dialog-box template.

Syntax

```
DLGTEMPLATE dialog-id [load-option] [mem-option]
BEGIN
dialog-definition
    .
    .
    .
END
```

Description

A dialog-box template consists of a series of statements that define the identifier, load and memory options, dialog-box dimensions, and controls in the dialog box. The dialog-box template can be loaded from the executable file by using the WinLoadDlg function.

You can provide any number of dialog-box templates in a resource script file, but each template must have a unique dialog-id value.

A DLGTEMPLATE statement can actually contain DIALOG, CONTROL, and WINDOW statements. Typically, you include only one DIALOG statement.

Example

This example uses a DLGTEMPLATE statement to create a dialog box.

```
DLGTEMPLATE ID_GETTIMER
BEGIN
    DIALOG "Timer", 1, 10, 10, 100, 40
    BEGIN
        LTEXT "Time (0 - 15):", 4, 8, 24, 72, 12
        ENTRYFIELD "0", ID_TIME, 80, 28, 16, 8, ES_MARGIN
        DEFPUSHBUTTON "Enter", ID_TIMEOK, 10, 6, 36, 12
        PUSHBUTTON "Cancel", ID_TIMECANCEL, 52, 6, 40, 12
    END
END
```

EDITTEXT Statement

The EDITTEXT statement creates an entry-field control.

Syntax

```
EDITTEXT text, id, x, y, width, height [, style]
```

Description

This control is a rectangle in which the user can type and edit text. The control displays a pointer when the user selects the control. The user can then use the keyboard to enter text or edit the existing text. Editing keys include the BACKSPACE and DELETE keys. By using the mouse or the DIRECTION keys, the user can select the characters to delete or select the place to insert new characters.

The EDITTEXT statement defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_ENTRYFIELD. If you do not specify a style, the default style is ES_AUTOSCROLL and WS_TABSTOP.

The EDITTEXT control statement is identical to the ENTRYFIELD control statement. Use the EDITTEXT statement only in a DIALOG or WINDOW statement.

Example

This example creates an entry-field control that is not labeled.

```
EDITTEXT "", 101, 10, 10, 24, 50
```

ENTRYFIELD Statement

The ENTRYFIELD statement creates an entry-field control.

Syntax

```
ENTRYFIELD text, id, x, y, width, height [, style]
```

Description

This control is a rectangle in which the user can type and edit text. The control displays a pointer when the user selects the control. The user can then use the keyboard to enter text or edit the existing text. Editing keys include the BACKSPACE and DELETE keys. By using the mouse or the DIRECTION keys, the user can select the characters to delete or select the place to insert new characters. The ENTRYFIELD statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_ENTRYFIELD. If you do not specify a style, the default style is ES_AUTOSCROLL and WS_TABSTOP.

Example

This example creates an entry-field control that is not labeled.

```
ENTRYFIELD "", 101, 10, 10, 24, 50
```

FONT Statement

The FONT statement defines a font resource for an application.

Syntax

```
FONT font-id [load-option] [mem-option] filename
```

Description

A font resource, typically created by using the OS/2 Font Editor, is a bit map defining the shape of the individual characters in a character set. The FONT statement copies the font resource from the file specified in the *filename* field and adds it to the other resources of the application. A font resource can be loaded from the executable file when needed by using the GpiLoadFonts function.

You can provide any number of FONT statements in a resource script file, but each statement must specify a unique font-id value.

Example

This example defines a font whose font identifier is 5. The font resource is copied from the file cmroman.fon.

```
FONT 5 cmroman.fon
```

FRAME Statement

The FRAME statement defines a frame window.

Syntax

```
FRAME text, id, x, y, width, height, style [, framectl]
    data-definitions
[ BEGIN
window-definition
    .
    .
    .
END ]
```

Description

The statement defines the title, identifier, position, and dimensions of the frame window, as well as the window style. The FRAME statement is most often used in a WINDOWTEMPLATE statement and, typically, only one FRAME statement is used. The FRAME statement, in turn, typically contains at least one WINDOW statement that defines the client window belonging to the frame window.

The frame window has no default style. You must use the *framectl* field to define additional frame controls, such as a title bar and system menu, to be created when the frame window is created. If the text field is not empty, the statement automatically adds a title-bar control to the frame window, whether or not you specify the FCF_TITLEBAR style. Frame controls are given default styles and control identifiers, depending on their class. For example, a title-bar control receives the identifier FID_TITLEBAR.

The FRAME statement can actually contain any combination of CONTROL, DIALOG, and WINDOW statements. Typically, a FRAME statement contains one WINDOW statement.

Example

This example creates a standard frame window with a title bar, a system menu, minimize and maximize boxes, and a vertical scroll bar. The FRAME statement contains a WINDOW statement defining the client window belonging to the frame window.

```
WINDOWTEMPLATE 1
BEGIN
    FRAME "My Window", 1, 10, 10, 320, 130, 0,
        FCF_STANDARD | FCF_VERTSCROLL
    BEGIN
        WINDOW "", FID_CLIENT, 0, 0, 0, 0, "MyClientClass"
    END
END
```

GROUPBOX Statement

The GROUPBOX statement creates a group-box control.

Syntax

GROUPBOX text, id, x, y, width, height [, style]

Description

The control is a rectangle that groups other controls together by drawing a border around them and displaying the given text in the upper-left corner.

The GROUPBOX statement defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_STATIC. If you do not specify a style, the default style is SS_GROUPBOX and WS_TABSTOP.

Use the GROUPBOX statement only in a DIALOG or WINDOW statement.

Example

This example creates a group-box control that is labeled "Options."

```
GROUPBOX "Options", 101, 10, 10, 100, 100
```

HELPITEM Statement

The HELPITEM statement defines the help items in a help table.

Syntax

```
HELPITEM application-window-id, help-subtable-id, extended-helppanel-id
```

Description

This statement specifies the resource identifier of an application window for which help is provided, along with the resource identifiers of the help subtable and extended help panel associated with the application window.

You can provide any number of HELPITEM statements in a HELPTABLE statement. You should provide one HELPITEM statement for each application window for which help is provided.

Use the HELPIITEM statement only in a HELPTABLE statement.

Example

This example defines a help item that associates a help subtable called IDSUB_FILEMENU and an extended help panel called IDEXT_APPHLP with an application window called IDWIN_FILEMENU.

```
HELPIITEM IDWIN_FILEMENU, IDSUB_FILEMENU, IDEXT_APPHLP
```

HELPSUBITEM Statement

The HELPSUBITEM statement defines the help subitems in a help subtable.

Syntax

```
HELPSUBITEM child-window-id, helppanel-id [, integer] ...
```

Description

This statement specifies the identifier of a child window for which help is provided, the identifier of the help panel associated with the child window, and one or more optional, application-defined integers.

You can provide any number of HELPSUBITEM statements in a HELPSUBTABLE statement. You should provide one HELPSUBITEM statement for each child window for which help is provided.

Use the HELPSUBITEM statement only in a HELPSUBTABLE statement.

Example

This example defines a help subitem that associates a child window called IDCLD_FILEMENU with a help panel called IDHP_FILEMENU.

```
HELPSUBITEM IDCLD_FILEMENU, IDHP_FILEMENU
```

HELPSUBTABLE Statement

The HELPSUBTABLE statement defines the contents of a help-subtable resource.

Syntax

```
HELPSUBTABLE helpsubtable-id
  [SUBITEMSIZE size]
BEGIN
  helpsubitem-definition
  .
  .
  .
END
```

Description

A help-subtable resource contains a help-subitem entry for each item that can be selected in an application window. Each of these items should be a child window of the application window specified in the help-table resource. The help subtable should contain a help subitem for each control, child window, and menu item in the application window.

You can provide any number of HELPSUBTABLE statements in a resource script file, but each statement must specify a unique helpsubtable-id value. You can also provide any number of helpsubitem-definition statements in the help subtable. These specify the child window for which help is provided, the help panel containing the help text for the child window, and one or more application-defined integers.

If you include optional integers in the helpsubitem-definition statements, you must also include a SUBITEMSIZE statement to specify the size, in words, of each help subitem. All help subitems in a help subtable must be the same size. The default size is two words per help subitem.

Example

This example creates a help-subtable resource whose help-subtable identifier is IDSUB_FILEMENU. Each HELPSUBITEM statement specifies a child window and a help panel.

```
HELPSUBTABLE IDSUB_FILEMENU
BEGIN
  HELPSUBITEM IDCLD_OPEN, IDPNL_OPEN
  HELPSUBITEM IDCLD_SAVE, IDPNL_SAVE
END
```

HELPTABLE Statement

The HELPTABLE statement defines the contents of a help-table resource.

Syntax

```
HELPTABLE helptable-id
BEGIN
helpitem-definition
.
.
.
END
```

Description

A help-table resource contains a help-item entry for each application window, dialog box, and message box for which help is provided.

You can provide any number of HELPTABLE statements in a resource script file, but each statement must specify a unique helptable-id value. You can also provide any number of helpitem-definition statements in the help table. These statements specify the application windows for which help is provided, the help subtables associated with each application window, and the extended help panels associated with each application window.

Example

This example creates a help-table resource whose help-table identifier is 1. Each HELPITEM statement specifies an application window, a help subtable, and an extended help panel.

```
HELPTABLE 1
BEGIN
    HELPITEM IDWIN_FILEMENU, IDSUB_FILEMENU, IDEXT_APPHLP
    HELPITEM IDWIN_EDITMENU, IDSUB_EDITMENU, IDEXT_APPHLP
END
```

ICON Statement (Resource)

This form of the ICON statement defines an icon resource for an application.

Syntax

```
ICON icon-id [load-option] [mem-option] filename
```

Description

An icon resource, typically created by using the Icon Editor, is a bit map defining the shape of the icon to be used for a given application. The ICON statement copies the icon resource from the file specified in the *filename* field and adds it to the application's other resources. An icon resource can be loaded when creating a window by using the WinCreateStdWindow function with the FS_ICON style.

You can provide any number of ICON statements in a resource script file, but each statement must specify a unique icon-id value.

An icon with an icon-id of 1 is the default icon. The RC program writes the icon not only to the resources in your executable file but also as the .ICON extended attribute.

Example

This example defines an icon whose icon identifier is 11. The icon resource is copied from the file custom.ico.

```
ICON 11 custom.ico
```

ICON Statement (Control)

This form of the ICON statement creates an icon control.

Syntax

```
ICON icon-id, id, x, y, width, height [, style]
```

Description

This control is an icon displayed in a dialog box. The ICON statement defines the icon-resource identifier, icon-control identifier, and position and attributes of a control window. The predefined class for this control is WC_STATIC. If you do not specify a style, the default style is SS_ICON. For the ICON statement, the *width* and *height* fields are ignored; the icon automatically sizes itself.

Use the ICON statement only in a DIALOG or WINDOW statement.

Example

This example creates an icon control whose icon identifier is 99.

```
ICON 99, 101, 10, 10, 0, 0
```

LISTBOX Statement

The LISTBOX statement creates commonly-used controls for a dialog box or window.

Syntax

```
LISTBOX id, x, y, width, height [, style]
```

Description

The control is a rectangle containing a list of user-selectable strings, such as file names.

The LISTBOX statement defines the identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_LISTBOX. If you do not specify a style, the default style is WS_TABSTOP.

Use the LISTBOX statement only in a DIALOG or WINDOW statement.

Example

This example creates a list-box control whose identifier is 101.

```
LISTBOX 101, 10, 10, 100, 100
```

LTEXT Statement

The LTEXT statement creates a left-aligned text control.

Syntax

```
LTEXT text, id, x, y, width, height [, style]
```

Description

The control is a simple rectangle displaying the given text left-aligned in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line. The LTEXT statement defines the text, identifier, dimensions, and attributes of the control. The predefined class for this control is WC_STATIC. If you do not specify a style, the default style is SS_TEXT, DT_LEFT, and WS_GROUP.

Use the LTEXT statement only in a DIALOG or WINDOW statement.

Example

This example creates a left-aligned text control that is labeled "Filename."

```
LTEXT "Filename", 101, 10, 10, 100, 100
```

MENU Statement

The MENU statement defines the contents of a menu resource.

Syntax

```
MENU menu-id [load-option] [mem-option]
BEGIN
menuitem-definition
.
.
.
END
```

Description

A menu resource is a collection of information that defines the appearance and function of an application menu. A menu is a special input tool that lets a user choose commands from a list of command names. A menu resource can be loaded from the executable file when needed by using the WinLoadMenu function.

You can provide any number of MENU statements in a resource script file, but each statement must specify a unique menu-id value. You can provide any number of menuitem-definition statements in the menu. These define the submenus and menu items (commands) in the menu. The order of the statements defines the order of the menu items.

Example

This example creates a menu resource whose menu identifier is 1. The menu contains a menu item named Alpha and a submenu named Beta. The submenu contains two menu items: Item 1 and Item 2.

```
=
MENU 1
BEGIN
    MENUITEM "Alpha", 100
    SUBMENU "Beta", 101
    BEGIN
        MENUITEM "Item 1", 200
        MENUITEM "Item 2", 201, , MIA_CHECKED
    END
END
```

MENUITEM Statement

The MENUITEM statement creates a menu item for a menu.

Syntax

```
MENUITEM text, menu-id [, menuitem-style [, menuitem-attributebrk.]
```

Description

This statement defines the text, identifier, and attributes of a menu item. Use the MENUITEM statement only in a MENU or SUBMENU statement.

The system displays the text when it displays the corresponding menu. If the user chooses the menu item, the system generates a WM_COMMAND message that includes the specified menu-item identifier and sends it to the window owning the menu.

You can provide any number of MENUITEM statements, but each must have a unique menu-id value.

The alternative form of the MENUITEM statement, MENUITEM SEPARATOR, creates a menu separator. A menu separator is a horizontal dividing bar between two menu items in a submenu. The separator is not active - that is, the user cannot choose it, it has no text associated with it, and it has no identifier.

You can use the \t or \a character combination in any item name. The \t character inserts a tab when the name is displayed and is typically used to separate the menu-item name from the name of an accelerator key. The \a character aligns to the right all text that follows it. These characters are intended to be used for menu items in submenus only. The width of the displayed submenu is always adjusted so there is at least one space (and usually more) between any pieces of text separated by a \t or \a. (When compiling the menu resource, the compiler stores the \t and \a characters as control characters. For example, the \t is stored as 0x09.)

A tilde (~) character in the item name indicates that the following character is used as a mnemonic character for the item. When the menu is displayed, the tilde is not shown, but the mnemonic character is underlined. The user can choose the menu item by pressing the key corresponding to the underlined mnemonic character.

Example

This example creates a menu item named Alpha. The item identifier is 101.

```
MENUITEM "Alpha", 101
```

This example creates a menu item named Beta. The item identifier is 102. The menu item has a text style and a checked attribute.

```
MENUITEM "Beta", 102, MIS_TEXT, MIA_CHECKED
```

This example creates a menu separator between menu items named Gamma and Delta.

```
MENUITEM "Gamma", 103  
MENUITEM SEPARATOR  
MENUITEM "Delta", 104
```

This example creates a menu item that has a bit map instead of a name. The bit-map identifier, 1, is first defined using a BITMAP statement. The identifier for the menu item is 301. Note that a # sign must be placed in front of the bit map identifier in the MENUITEM statement.

```
BITMAP 1 mybitmap.bmp  
  
MENUITEM "#1", 301, MIS_BITMAP
```

MESSAGETABLE Statement

The MESSAGETABLE statement creates one or more string resources for an application.

Syntax

```
MESSAGETABLE [load-option] [mem-option]  
BEGIN  
string-id string-definition  
.  
.  
.  
END
```

Description

A string resource is a null-terminated character string that has a unique string identifier. A string resource can be loaded from the executable file when needed by using the DosGetResource function with the RT_MESSAGE resource type. RT_MESSAGE resources are bundled together in groups of 16, with any missing IDs replaced with zero length strings. Each group, or bundle, is assigned a unique sequential ID. The resource string ID is not necessarily the same as the ID specified when using DosGetResource. The formula for calculating the ID of the

resource bundle, for use in DosGetResource, is as follows:

```
bundle ID = ( id / 16 ) + 1,
```

where id is the string ID assigned in the RC file.

Thus, bundle 1 contains strings 0 to 15, bundle 2 contains strings 16 to 31, and so on. Once the address of the bundle has been returned by DosGetResource (using the calculated ID), the buffer can be parsed to locate the particular string within the bundle. The number of the string is calculated by the formula:

```
string = id % 16
```

(string = remainder for id/16).

The buffer returned consists of the CodePage of the strings in the first USHORT, followed by the 16 strings in the bundle. The first BYTE of each string is the length of the string (including the null terminator), followed by the string and the terminator. A zero length string is represented by two bytes: 01 (string length) followed by the null terminator.

You can provide any number of MESSAGETABLE statements in a resource script file. The compiler treats all the strings from the various MESSAGETABLE statements as if they belonged to a single statement. This means that no two strings in a resource script file can have the same string identifier.

Although the MESSAGETABLE and STRINGTABLE statements are nearly identical, most applications use the STRINGTABLE statement instead of the MESSAGETABLE statement to create string resources.

You can continue a string on multiple lines by terminating the line with a backslash (\) or by terminating the line with a double quotation mark (") and then starting the next line with a double quotation mark.

Example

This example creates two string resources whose string identifiers are 1 and 2.

```
MESSAGETABLE
BEGIN
    1 "Filename not found"
    2 "Cannot open file for reading"
END
```

MLE Statement

The MLE statement creates a multiple-line entry-field control.

Syntax

```
MLE text, id, x, y, width, height [, style]
```

Description

The control is a rectangle in which the user can type and edit multiple lines of text. The control displays a pointer when the user selects it. The user can then use the keyboard to enter text or edit the existing text. Editing keys include the BACKSPACE and DELETE keys. By using the mouse or the DIRECTION keys, the user can select the characters to delete or select the place to insert new characters. The MLE statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_MLE. If you do not specify a style, the default style is MLS_BORDER, WS_GROUP, and WS_TABSTOP.

Example

This example creates a multiple-line entry-field control that is not labeled.

```
MLE "", 101, 10, 10, 50, 100
```

NOTEBOOK Statement

The NOTEBOOK statement creates a notebook control within the dialog window.

Syntax

```
NOTEBOOK id, x, y, width, height [, style]
```

Description

This control is used to organize information on individual pages so that it can be located and displayed easily. The NOTEBOOK statement defines the identifier, position, dimensions, and attributes of a notebook control. The predefined class for this control is WC_NOTEBOOK. If you do not specify a style, the default style is WS_TABSTOP and WS_VISIBLE.

Use the NOTEBOOK statement only in a DIALOG or WINDOW statement.

Example

This example creates a notebook control at position (20, 20) within the dialog window. The notebook has a width of 200 character units and a height of 50 character units. Its resource ID is 201. The tabs style BKS_ROUNDEDTABS specification overrides the notebook default style of square tabs. The default styles WS_TABSTOP and WS_GROUP are both in effect, although only the latter is specified.

```
#define IDC_NOTEBOOK 201
#define IDD_NOTEBOOKDLG 503
DIALOG "Notebook", IDD_NOTEBOOKDLG, 11, 11, 420, 420, FS_NOBYTEALIGN |
    WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
```

```
BEGIN
  NOTEBOOK    IDC_NOTEBOOK, 20, 20, 200, 400, BKS_ROUNDED TABS | WS_GROUP
END
```

POINTER Statement

The POINTER statement defines a pointer resource for an application.

Syntax

```
POINTER pointer-id [load-option] [mem-option] filename
```

Description

A pointer resource, typically created by using the OS/2 Icon Editor, is a bit map defining the shape of the mouse pointer on the screen. The POINTER statement copies the pointer resource from the file specified in the *filename* field and adds it to the application's other resources. A pointer resource can be loaded from the executable file when needed by using the WinLoadPointer function.

You can provide any number of POINTER statements in a resource script file, but each statement must specify a unique pointer-id value.

Example

This example defines a pointer whose pointer identifier is 10. The pointer resource is copied from the file custom.cur.

```
POINTER 10 custom.cur
```

PRESPARAMS Statement

The PRESPARAMS statement defines presentation fields that customize a dialog box, menu, window, or control.

Syntax

```
PRESPARAMS presparam, value [, value] ...
```

Description

PRESPARAMS data is a series of types and values. The window procedure of the dialog box, menu, window, or control receives and processes this data when the item is created. The data for custom controls can be in any format.

PRESPARAMS is often used to supply data to control the appearance of the customized window when it is first created. For example, the PRESPARAMS statement may specify the colors to be used in the window.

Example

This example creates a menu resource with a menu identifier of 1. The PRESPARAMS statement specifies that the following three menu items be displayed in the 12-point Helvetica** font.

```
MENU 1
BEGIN
    PRESPARAMS PP_FONTNAMESIZE, "12.Helv"
    MENUITEM "New", 100
    MENUITEM "Open", 101
    MENUITEM "Save", 102
END
```

PUSHBUTTON Statement

The PUSHBUTTON statement creates a push-button control.

Syntax

```
PUSHBUTTON text, id, x, y, width, height [, style ]
```

Description

The control is a round-cornered rectangle containing the given text. The control sends a message to its parent whenever the user chooses the control. The PUSHBUTTON statement defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_BUTTON. If you do not specify a style, the default style is BS_PUSHBUTTON and WS_TABSTOP.

Use the PUSHBUTTON statement only in a DIALOG or WINDOW statement.

Example

This example creates a push-button control that is labeled "OK."

```
PUSHBUTTON "OK", 101, 10, 10, 100, 100
```

RADIOBUTTON Statement

The RADIOBUTTON statement creates a radio-button control, which is a small circle that has the given text displayed to its right.

Syntax

```
RADIOBUTTON text, id, x, y, width, height [, style]
```

Description

The control highlights the circle and sends a message to its parent window when the user selects the button. The control removes the highlight and sends a message when the button is next selected. The RADIOBUTTON statement defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_BUTTON. If you do not specify a style, the default style is BS_RADIOBUTTON.

Use the RADIOBUTTON statement only in a DIALOG or WINDOW statement.

Example

This example creates a radio-button control that is labeled "Italic."

```
RADIOBUTTON "Italic", 101, 10, 10, 24, 50
```

RCDATA Statement

The RCDATA statement defines a custom-data resource for an application.

Syntax

```
RCDATA resource-id
BEGIN
data-definition [, data-definition] ...
.
.
.
END
```

Description

The custom data can be in whatever format the application requires. You can provide any number of RCDATA statements in a resource script file, but each statement must specify a unique resource-id value. A custom-data resource can be loaded from the executable file when needed by using the `DosGetResource` or `DosGetResource2` functions with the `RT_RCDATA` resource type.

Example

This example defines custom data that has a resource identifier of 5.

```
RCDATA 5
BEGIN
    "E. A. Poe", 1849, -32, 3L, 0x80000001, 3+4+5
END
```

RCINCLUDE Statement

The `RCINCLUDE` statement causes RC to process the resource script file specified in the *filename* field along with the current resource script file.

Syntax

```
RCINCLUDE filename
```

Description

The contents of both script files are compiled by RC and the results are placed in one binary resource file and/or executable file.

RCINCLUDE statements are processed before any other processing is done, including preprocessing by RCPP.EXE, which removes comments, replaces values in the define directives, and so forth.

When specifying a high performance file system (HPFS) file name on an RCINCLUDE statement, enclose the path and file name in double quotes; for example:

```
RCINCLUDE "d:\project\long dialog.dlg"
```

Double quotes enable the resource compiler to recognize a name containing embedded blank characters.

Example

This example includes the file DIALOGS.RC as part of the current resource script file.

```
RCINCLUDE dialogs.rc
```

RESOURCE Statement

The RESOURCE statement defines a custom resource for an application.

Syntax

```
RESOURCE type-id resource-id [load-option] [mem-option] filename
```

Description

A custom resource can be any data in any format. The RESOURCE statement copies the custom resource from the specified file and adds it to the application's other resources. A custom resource can be loaded from the executable file when needed by using the DosGetResource or DosGetResource2 function and specifying the resource's type and resource identifier.

You can provide any number of RESOURCE statements in a resource script file, but each statement must specify a unique combination of type-id and resource-id values. That is, RESOURCE statements having the same type-id value are permitted as long as the resource-id value for each is unique.

Example

This example defines a custom resource whose type identifier is 300 and whose resource identifier is 14. The custom resource is copied from the file CUSTOM.RES.

```
RESOURCE 300 14 custom.res
```

RTEXT Statement

The RTEXT statement creates a right-aligned text control.

Syntax

```
RTEXT text, id, x, y, width, height [, style]
```

Description

The control is a simple rectangle displaying the given text right-aligned in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line. The RTEXT statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of the control. The predefined class for the control is WC_STATIC. If you do not specify a style, the default style is SS_TEXT, DT_RIGHT, and WS_GROUP.

Example

This example creates a right-aligned text control that is labeled "Filename."

```
RTEXT "Filename", 101, 10, 10, 100, 100
```

SLIDER Statement

The SLIDER statement creates a slider control within the dialog window.

Syntax

SLIDER id, x, y, width, height [, style]

Description

This control lets the user set, display, or modify a value by moving a slider arm along a slider shaft. The SLIDER statement defines the identifier, position, dimensions, and attributes of a slider control. The predefined class for this control is WC_SLIDER. If you do not specify a style, the default style is WS_TABSTOP and WS_VISIBLE.

Use the SLIDER statement only in a DIALOG or WINDOW statement.

Example

This example creates a slider control at position (40, 30) within the dialog window. The slider has a width of 120 character units and a height of 2 character units. Its resource ID is 101. The style specification SLS_BUTTONSLEFT adds buttons to the left of the slider shaft. The default styles WS_TABSTOP and WS_VISIBLE are both in effect, although only the latter is specified.

```
#define IDC_SLIDER 101
#define IDD_SLIDERDLG 502
DIALOG "Slider", IDD_SLIDERDLG, 11, 11, 200, 240, FS_NOBYTEALIGN |
    WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
BEGIN
    SLIDER IDC_SLIDER, 40, 30, 120, 16, SLS_BUTTONSLEFT | WS_VISIBLE
END
```

SPINBUTTON Statement

The SPINBUTTON statement creates a spin-button control within the dialog window.

Syntax

SPINBUTTON id, x, y, width, height [, style]

Description

This control gives the user quick access to a finite set of data. The SPINBUTTON statement defines the identifier, position, dimensions, and attributes of a spin-button control. The predefined class for this control is WC_SPINBUTTON. If you do not specify a style, the default style is WS_TABSTOP, WS_VISIBLE, and SPBS_MASTER.

Use the SPINBUTTON statement only in a DIALOG or WINDOW statement.

Example

This example creates a spin-button control at position (80, 20) within the dialog window. The spin button has a width of 60 character units and a height of 3 character units. Its resource ID is 302. The style specification SPBS_NUMERICONLY creates a control which accepts only the digits 0-9 and virtual keys. The default styles SPBS_MASTER, WS_TABSTOP, and WS_VISIBLE are all in effect, although only WS_TABSTOP is specified.

```
#define IDC_SPINBUTTON 302
#define IDD_SPINDLG 502
DIALOG "Spin button", IDD_SPINDLG, 11, 11, 200, 240, FS_NOBYTEALIGN |
    WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
BEGIN
    SPINBUTTON IDC_SPINBUTTON, 80, 20, 60, 24, SPBS_NUMERICONLY | WS_TABSTOP
END
```

STRINGTABLE Statement

The STRINGTABLE statement creates one or more string resources for an application.

Syntax

```
STRINGTABLE [load-option] [mem-option]
BEGIN
string-id string-definition
.
.
.
END
```

Description

A string resource is a null-terminated character string that has a unique string identifier. A string resource can be loaded from the executable file when needed by using WinLoadString or with DosGetResource with the RT_STRING resource type. RT_STRING resources are bundled together in groups of 16, with any missing IDs replaced with zero length strings. Each group, or bundle, is assigned a unique sequential ID. The resource string ID is not necessarily the same as the ID specified when using DosGetResource. The formula for calculating the ID of the resource bundle, for use in DosGetResource, is as follows:

$$\text{bundle ID} = (\text{id} / 16) + 1$$

where id is the string ID assigned in the RC file.

Thus, bundle 1 contains strings 0 to 15, bundle 2 contains strings 16 to 31, and so on. Once the address of the bundle has been returned by DosGetResource (using the calculated ID), the buffer can be parsed to locate the particular string within the bundle. The number of the string is calculated by the formula:

```
string = id % 16
```

(string = remainder for id/16).

The buffer returned consists of the CodePage of the strings in the first USHORT, followed by the 16 strings in the bundle. The first BYTE of each string is the length of the string (including the null terminator), followed by the string and the terminator. A zero length string is represented by two bytes: 01 (string length) followed by the null terminator.

You can provide any number of STRINGTABLE statements in a resource script file. The compiler treats all the strings from the various STRINGTABLE statements as if they belonged to a single statement. This means that no two strings in a resource script file can have the same string identifier.

You can continue a string on multiple lines by terminating the line with a backslash (\) or by terminating the line with a double quotation mark (") and then starting the next line with a double quotation mark.

Example

This example creates two string resources whose string identifiers are 1 and 2.

```
#define IDS_HELLO    1
#define IDS_GOODBYE  2

STRINGTABLE
BEGIN
    IDS_HELLO    "Hello"
    IDS_GOODBYE  "Goodbye"
END
```

SUBITEMSIZE Statement

The SUBITEMSIZE statement specifies the size, in words, of each help subitem in a help subtable.

Syntax

```
SUBITEMSIZE  size
```

Description

The minimum size is two words, and each help subitem in a help subtable must be the same size. When used, the SUBITEMSIZE statement must appear after the HELPSUBTABLE statement and before the BEGIN keyword.

You do not need to use the SUBITEMSIZE statement if the help subitems are the default size (2).

Example

The SUBITEMSIZE statement in this example specifies that each HELPSUBITEM statement contains three words.

```
HELPSUBTABLE 1
SUBITEMSIZE 3
BEGIN
    HELPSUBITEM IDCLD_FILEMENU, IDHP_FILEMENU, 5
    HELPSUBITEM IDCLD_HELPMENU, IDHP_HELPMENU, 6
END
```

SUBMENU Statement

The SUBMENU statement creates a submenu for a given menu.

Syntax

```
SUBMENU text, submenu-id [, menuitem-style[, menuitem-attributerbrk.]]
BEGIN
menuitem-definition
    .
    .
    .
END
```

Description

A submenu is a vertical list of menu items from which the user can choose a command.

You can provide any number of SUBMENU statements in a MENU statement, but each SUBMENU statement must specify a unique submenu-id value. You can provide any number of menuitem-definition statements in the SUBMENU statement. These define the menu items (commands) in the menu. The order of the statements determines the order of the menu items.

Example

This example creates a submenu named Elements. Its identifier is 2. The submenu contains three menu items, which are created by using MENUITEM statements.

```
SUBMENU "Elements", 2
BEGIN
    MENUITEM "Oxygen", 200
    MENUITEM "Carbon", 201, , MIA_CHECKED
    MENUITEM "Hydrogen", 202
END
```

VALUESET Statement

The VALUESET statement creates a value set control within the dialog window.

Syntax

```
VALUESET    id, x, y, width, height [, style]
```

Description

This control lets a user select one choice from a group of mutually exclusive choices. The VALUESET statement defines the identifier, position, dimensions, and attributes of a value set control. The predefined class for this control is WC_VALUESET. If you do not specify a style, the default style is WS_TABSTOP and WS_VISIBLE.

Use the VALUESET statement only in a DIALOG or WINDOW statement.

Example

This example creates a value set control at position (40, 40) within the dialog window. The value set control has a width of 220 character units and a height of 20 character units. Its resource ID is 302. The style specification VS_ICON creates a control to show items in icon form. The default styles WS_TABSTOP and WS_VISIBLE are both in effect, although only WS_TABSTOP is specified.

```
#define      IDC_VALUESET      302
#define      IDD_VALUESETDLG   501
DIALOG "Value set", IDD_VALUESETDLG, 11, 11, 260, 240, FS_NOBYTEALIGN |
    WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
BEGIN
    VALUESET  IDC_VALUESET, 40, 40, 220, 160, VS_ICON | WS_TABSTOP
END
```

WINDOW Statement

The WINDOW statement creates a window of the specified class.

Syntax


```

WINDOW text, id, x, y, width, height, class [, style [, framect]]
  data-definitions
[ BEGIN
control-definition
  .
  .
  .
END ]

```

Description

The statement defines the position and dimensions of the window relative to its parent window, as well as the window-box style. The WINDOW statement is typically used in a WINDOWTEMPLATE or FRAME statement.

Usually, only one WINDOW statement is used in a FRAME statement. It defines the client window belonging to the corresponding frame window. The optional BEGIN and END keywords enclose any CONTROL statements that are given with the window. CONTROL statements given in this manner represent child windows belonging to the window created by the WINDOW statement.

The WINDOW statement can actually contain any combination of CONTROL, DIALOG, and WINDOW statements. Typically, a WINDOW statement contains one or no such statements.

Example

This example creates a client window belonging to the frame window. The client window belongs to the "MyClientClass" window class and has the standard window identifier FID_CLIENT.

```

WINDOWTEMPLATE 1
BEGIN
  FRAME "My Window", 1, 10, 10, 320, 130,
    0, FCF_STANDARD | FCF_VERTSCROLL
  BEGIN
    WINDOW "", FID_CLIENT, 0, 0, 0, 0, "MyClientClass"
  END
END

```

WINDOWTEMPLATE Statement

The WINDOWTEMPLATE statement creates a window template.

Syntax

```

WINDOWTEMPLATE window-id [load-option] [mem-option]
BEGIN
window-definition
  .
  .

```

END

Description

A window template consists of a series of statements that define the window identifier, load and memory options, window dimensions, and controls in the window. The window template can be loaded from the executable file by using the WinLoadDlg function.

You can provide any number of window templates in a resource script file, but each template must have a unique window-id value.

A WINDOWTEMPLATE statement can contain DIALOG, CONTROL, and WINDOW statements. Typically, only one WINDOW statement is used in the WINDOWTEMPLATE statement.

Directive Descriptions

This section provides the syntax, a description, and an example of each of the directives.

#define Directive

The #define directive assigns the given value to the specified name. All subsequent occurrences of the name are replaced by the value.

Syntax

```
#define name value
```

Example

This example assigns values to the names "NONZERO" and "USERCLASS".

```
#define NONZERO 1
#define USERCLASS "MyControlClass"
```

#elif Directive

The `#elif` directive marks an optional clause of a conditional-compilation block defined by a `#ifdef`, `#ifndef`, or `#if` directive. The directive controls conditional compilation of the resource file by checking the specified constant expression. If the constant expression is nonzero, `#elif` directs the compiler to continue processing statements up to the next `#endif`, `#else`, or `#elif` directive and then skip to the statement after `#endif`. If the constant expression is zero, `#elif` directs the compiler to skip to the next `#endif`, `#else`, or `#elif` directive. You can use any number of `#elif` directives in a conditional block.

Syntax

```
#elif constant-expression
```

Example

In this example, `#elif` directs the compiler to process the second `BITMAP` statement only if the value assigned to the name "Version" is less than 7. The `#elif` directive itself is processed only if Version is greater than or equal to 3.

```
#if Version < 3
BITMAP 1 errbox.bmp
#elif Version < 7
BITMAP 1 userbox.bmp
#endif
```

#else Directive

The `#else` directive marks an optional clause of a conditional-compilation block defined by a `#ifdef`, `#ifndef`, or `#if` directive. The `#else` directive must be the last directive before the `#endif` directive.

This directive has no arguments.

Syntax

```
#else
```

Example

This example compiles the second `BITMAP` statement only if the name "DEBUG" is not defined.

```
#ifdef DEBUG
    BITMAP 1 errbox.bmp
#else
    BITMAP 1 userbox.bmp
#endif
```

#endif directive

The `#endif` directive marks the end of a conditional-compilation block defined by a `#ifdef` directive. One `#endif` is required for each `#if`, `#ifdef`, or `#ifndef` directive.

This directive has no arguments.

Syntax

```
#endif
```

#if Directive

The `#if` directive controls conditional compilation of the resource file by checking the specified constant expression. If the constant expression is nonzero, `#if` directs the compiler to continue processing statements up to the next `#endif`, `#else`, or `#elif` directive and then skip to the statement after the `#endif` directive. If the constant expression is zero, it directs the compiler to skip to the next `#endif`, `#else`, or `#elif` directive.

Syntax

```
#if constant-expression
```

Example

This example compiles the `BITMAP` statement only if the value assigned to the name "Version" is less than 3.

```
#if Version < 3
BITMAP 1 errbox.bmp
#endif
```

#ifdef Directive

The `#ifdef` directive controls conditional compilation of the resource file by checking the specified name. If the name has been defined by using a `define` directive or by using the `-d` command-line option of `rc`, `#ifdef` directs the compiler to continue with the statement immediately after the `#ifdef` directive. If the name has not been defined, `#ifdef` directs the compiler to skip all statements up to the next `#endif` directive.

Syntax

`#ifdef name`

Example

This example compiles the `BITMAP` statement only if the name "Debug" is defined.

```
#ifdef Debug
BITMAP 1 errbox.bmp
#endif
```

#ifndef Directive

The `#ifndef` directive controls conditional compilation of the resource file by checking the specified name. If the name has not been defined or if its definition has been removed by using the `#undef` directive, `#ifndef` directs the compiler to continue processing statements up to the next `#endif`, `#else`, or `#elif` directive and then skip to the statement after the `#endif` directive. If the name is defined, `#ifndef` directs the compiler to skip to the next `#endif`, `#else`, or `#elif` directive.

Syntax

`#ifndef name`

Example

This example compiles the `BITMAP` statement only if the name "Optimize" is not defined.

```
#ifndef Optimize
BITMAP 1 errbox.bmp
#endif
```

#include Directive

The `#include` directive causes RC to process the file specified in the *filename* field. This file should be a header file that defines the constants used in the resource script file. Only the define directives in the specified file are processed. All other statements are ignored.

The *filename* field is handled as a C string. Therefore, you must include two backslashes (\\) wherever one is required in the path. (As an alternative, you can use a single forward slash (/) instead of two backslashes.)

Syntax

```
#include filename
```

Example

This example processes the header files OS2.H and HEADERS\MYDEFS.HI while compiling the resource script file.

```
#include <os2.h>
#include "headers\\mydefs.h"
```

#undef Directive

The `undef` directive removes the current definition of the specified name. All subsequent occurrences of the name are processed without replacement.

Syntax

```
#undef name
```

Example

This example removes the definitions for the names "nonzero" and "USERCLASS".

```
#undef      nonzero
#undef      USERCLASS
```

Using Resource Files

This section explains how to create a resource script file, compile it using the Resource Compiler (RC.EXE), and optionally add the resources to your executable file. Resource script files have a default file-name extension of .RC.

For resource information on the individual controls, see the chapter on the specific control. For example, an example of a resource script file for frame windows is in [Frame Windows](#).

Creating and Compiling a Resource File

The resource compiler (RC) compiles a resource script file to create a new file, called a binary resource file, which has a .RES file-name extension. The binary resource file can be added to the executable file of the application, thereby replacing any existing resources in that file.

The RC command line has the following three basic forms:

```
rc resource-script-file [executable-file]

rc binary-resource-file [executable-file]

rc -r resource-script-file [binary-resource-file]
```

Note: The third option does not add to the executable file.

The *resource-script-file* parameter is the file name of the resource script file to be compiled.

The *executable-file* parameter must be the name of the executable file to receive the compiled resources. This is a file having a file-name extension of either .EXE or .DLL. If you omit the executable-file field, RC adds the compiled resources to the executable file that has the same name as the resource script file but which has the .EXE file-name extension.

The *binary-resource-file* parameter is the name of the binary resource file to be added to the executable file.

The *-r* option directs RC to compile the resource script file without adding it to an executable file.

Compiling and Adding Resources to the .EXE File

To compile the resource script file EXAMPLE.RC and add the result to the executable file EXAMPLE.EXE, use the following command:

```
rc example
```

You do not need to specify the .RC extension. RC creates the binary resource file EXAMPLE.RES and adds the compiled resource to the executable file EXAMPLE.EXE.

Compiling without Adding Resources to the .EXE File

To compile the resource script file EXAMPLE.RC into a binary resource file without adding the resources to an executable file, use the following command:

```
rc -r example
```

The compiler creates the binary resource file EXAMPLE.RES. To create a binary resource file that has a name different from the resource script file, use the following command:

```
rc -r example newfile.res
```

Adding the Compiled Resources to the .EXE File

To add the compiled resources in the binary resource file EXAMPLE.RES to an executable file, use the following command:

```
rc example.res
```

To specify the name of the executable file, if the name is different from the resource file, use the following command:

```
rc example.res newfile.exe
```

Adding the Compiled Resources to a DLL

To add the compiled resources to a dynamic-link-library (DLL) file, use the following command:

```
rc example.res dynalink.dll
```

Scroll-Bar Controls

Scroll bars are control windows that convert mouse and keyboard input into integers; they are used by an application to scroll the contents of a client window. This chapter describes how to create and use scroll bars in PM applications.

About Scroll Bars

A scroll bar has three main parts: the bar, its arrows, and a slider.

The arrows are located at each end of the scroll bar. The left scroll arrow, on the left side of a horizontal scroll bar, enables the user to scroll to the left in a document. The right scroll arrow lets the user scroll to the right.

On a vertical scroll bar, the upper scroll arrow enables the user to scroll upward in the document; the lower scroll arrow, downward. The slider, which lies between the two scroll arrows, reflects the current value of the scroll bar. Scroll bars monitor the slider and send notification messages to the owner window when the slider position changes as a result of mouse or keyboard input.

Although, typically, scroll bars are used in frame windows, an application can use stand-alone scroll bars of any size or shape, at any position, in a window of almost any class. Scroll bars can be used as parts of other control windows; for example, a list box uses a scroll bar to enable the user to view items when the list box is too small to display all the items.

Scroll-Bar Creation

An application can include a scroll bar in a standard frame window by specifying the FCF_HORZSCROLL or FCF_VERTSCROLL flag in the WinCreateStdWindow function. To create a scroll bar in another type of window, an application can specify the predefined (preregistered) window class WC_SCROLLBAR in the WinCreateWindow function or in the CONTROL statement in a resource file.

Although most applications specify an owner window when creating a scroll bar, an owner is not required. If an application does not specify an owner, the scroll bar does not send notification messages.

Scroll-Bar Styles

A scroll bar has styles that determine what it looks like and how it responds to input. Styles are specified in the WinCreateWindow function or the CONTROL statement. A scroll-bar can have the following styles:

Style	Meaning
SBS_AUTOTRACK	Causes the entire slider to track the movement of the mouse pointer when the user scrolls the window. Without this style, only an outlined image of the slider tracks the movement of the mouse pointer, and the slider jumps to the new location when the user releases the mouse button.
SBS_HORZ	Creates a horizontal scroll bar.
SBS_THUMBSize	Used to calculate the size of the scroll-bar slider from the SBCDATA passed to WinCreateWindow.
SBS_VERT	Creates a vertical scroll bar.

Scroll-Bar Range and Position

Every scroll bar has a range and a slider position. The range specifies the minimum and maximum values for the slider position. As the user moves the slider in a scroll bar, the scroll bar reports the slider position as an integer in this range. If the slider position is the minimum value, the slider is at the top of a vertical scroll bar or at the left end of a horizontal scroll bar. If the slider position is the maximum value, the slider is at the bottom or right end of the vertical or horizontal scroll bar, respectively.

An application can adjust the range to convenient integers by using SBM_SETSCROLLBAR or WM_SETWINDOWPARAMS, or by using the SBCDATA structure during creation of the scroll bar. This enables you to easily translate the slider position into a value that corresponds to the data being scrolled. For example, an application attempting to display 100 lines of text (numbered 0 to 99) in a window that can show only 20 lines at a time could set the vertical scroll-bar range from 0-99, permitting any line to be the top line, and requiring blank lines to fill the viewing area when there are not sufficient lines of information to fill the area (lines 80-99). More likely, the range would be set to 0-79, so that only the first 80 lines could be the top line; this guarantees that there would always be 20 lines of text to fill the window.

The current settings can be obtained using SBM_QUERYRANGE or WM_QUERYWINDOWPARAMS.

To establish a useful relationship between the scroll-bar range and the data, an application must adjust the range whenever the data or the size of the window changes. This means the application should adjust the range as part of processing WM_SIZE messages.

An application must move the slider in a scroll bar. Although the user requests scrolling in a scroll bar, the scroll bar does not update the slider position. Instead, it passes the request to the owner window, which scrolls the data and updates the slider position using the SBM_SETPOS message. The application controls the slider movement and can move the slider in the increments best suited for the data being scrolled.

An application can retrieve the current slider position of a scroll bar by sending the SBM_QUERYPOS message to the scroll bar.

If a scroll bar is a descendant of a frame window, its position relative to its parent can change when the position of the frame window changes. Frame windows draw scroll bars relative to the upper-left corner of the frame window (rather than the lower-left corner). The frame window can adjust the *y* coordinate of the scroll-bar position, which would be desirable if the scroll bar is a child of the frame window, but would be undesirable if the scroll bar is not a child window.

Scroll-Bar Slider Size

The slider can be displayed either as a square (the default), or as a portion of the scroll bar if SBCDATA and the SBS_THUMBSize style are specified at creation. Displaying the slider as a proportional rectangle permits the size of the slider to be proportional to the amount of data being viewed in the visible range. The size is set based on the visible range and the number of values in the range. As an example, where the viewing area is 20 items and the range is 100, the slider size would be 20% of the potential slider area. Note that there is no direct connection between the scroll bar range and the range value used to set the slider size. It is possible to set the scroll-bar range from 0-99, and base the slider size on a viewing area of 500 and a range of 1000. This will set the scroll-bar to have 100 positions and will display a slider that is half the size of the scroll bar.

The slider size can be set using SBM_SETTHUMBSize or WM_SETWINDOWPARAMS, and obtained using WM_QUERYWINDOWPARAMS.

Scroll-Bar Notification Messages

A scroll bar sends notification messages to its owner whenever the user clicks the scroll bar. WM_VSCROLL and WM_HSCROLL are the notification messages for vertical and horizontal scroll bars, respectively. If the scroll bar is a frame control window, the frame window passes the message to its client window.

Each notification message includes the scroll-bar identifier, scroll-bar command code corresponding to the action of the user, and, in some cases, the position of the slider. If an application creates a scroll bar as part of a frame control window, the scroll-bar identifier is the predefined constant FID_VERTSCROLL or FID_HORZSCROLL. Otherwise, it is the identifier given in the WinCreateWindow function.

The scroll-bar command codes specify the action the user has taken. Operating system user-interface guidelines recommend certain responses for each action.

Following is a list of the command codes; for each code, the user action is specified, followed by the application's response. In each case, a scrolling unit, appropriate for the given data, must be defined by the application. For example, for scrolling text vertically, the typical unit is a line.

Command Code	Description
SB_LINEUP	Indicates that the user clicked the top scroll arrow. Decrement the slider position by one, and scroll toward the top of the data by one unit.
SB_LINEDOWN	Indicates that the user clicked the bottom scroll arrow. Increment the slider position by one, and scroll toward the bottom of the data by one unit.
SB_LINELEFT	Indicates that the user clicked the left scroll arrow. Decrement the slider position by one, and scroll toward the left end of the data by one unit.

<code>SB_LINERIGHT</code>	Indicates that the user clicked the right scroll arrow. Increment the slider position by one, and scroll toward the right end of the data by one unit.
<code>SB_PAGEUP</code>	Indicates that the user clicked the scroll-bar background above the slider. Decrement the slider position by the number of data units in the window, and scroll toward the top of the data by the same number of units.
<code>SB_PAGEDOWN</code>	Indicates that the user clicked the scroll-bar background below the slider. Increment the slider position by the number of data units in the window, and scroll toward the bottom of the data by the same number of units.
<code>SB_PAGELEFT</code>	Indicates that the user clicked the scroll-bar background to the left of the slider. Decrement the slider position by the number of data units in the window, and scroll toward the left end of the data by the same number of units.
<code>SB_PAGERIGHT</code>	Indicates that the user clicked the scroll-bar background to the right of the slider. Increment the slider position by the number of data units in the window, and scroll toward the right end of the data by the same number of units.
<code>SB_SLIDERTRACK</code>	Indicates that the user is dragging the slider. Applications that draw data quickly can set the slider to the position given in the message, and scroll the data by the same number of units the slider has moved. Applications that cannot draw data quickly should wait for the <code>SB_SLIDERPOSITION</code> code before moving the slider and scrolling the data.
<code>SB_SLIDERPOSITION</code>	Indicates that the user released the slider after dragging it. Set the slider to the position given in the message, and scroll the data by the same number of units the slider was moved.
<code>SB_ENDSCROLL</code>	Indicates that the user released the mouse after holding it on an arrow or in the scroll-bar background. No response is necessary.

If the command code is `SB_SLIDERTRACK` or `SB_SLIDERPOSITION`, indicating that the user is moving the scroll-bar slider, the notification message also contains the current position of the slider.

The owner window can send a message to the scroll bar to read or reset the current value and range of the scroll bar. To reflect any changes in the state of the scroll bar, the owner window also can adjust the data the scroll bar controls.

An application can use the `WinEnableWindow` function to disable a scroll bar. A disabled scroll bar ignores the actions of the user, sending out no notification messages when the user tries to manipulate it. If an application has no data to scroll, or if all data fits in the client window, the application should disable the scroll bar.

Scroll Bars and the Keyboard

When a scroll bar has the keyboard focus, it generates notification messages for the following keys:

Keys	Response
UP	SB_LINEUP or SB_LINELEFT
LEFT	SB_LINEUP or SB_LINELEFT
DOWN	SB_LINEDOWN or SB_LINERIGHT
RIGHT	SB_LINEDOWN or SB_LINERIGHT
PGUP	SB_PAGEUP or SB_PAGELEFT
PGDN	SB_PAGEDOWN or SB_PAGERIGHT

If an application uses scroll bars to scroll data but does not give the scroll bar the input focus, the window with the focus must process keyboard input. The window can generate scroll-bar notification messages or carry out the indicated scrolling. The following table shows the responses to keys that a window must process:

Key	Response
UP	SB_LINEUP
DOWN	SB_LINEDOWN
PGUP	SB_PAGEUP
PGDN	SB_PAGEDOWN
CTRL+HOME	SB_SLIDERTRACK, with the slider set to the minimum position
CTRL+END	SB_SLIDERTRACK, with the slider set to the maximum position
LEFT	SB_LINELEFT
RIGHT	SB_LINERIGHT
CTRL+PGUP	SB_PAGELEFT
CTRL+PGDN	SB_PAGERIGHT
HOME	SB_SLIDERTRACK, with the slider set to the minimum position
END	SB_SLIDERTRACK, with the slider set to the maximum position

For vertical scroll bars that are part of list boxes, the following table shows the responses to keys:

Key	Command
CTRL+UP	SB_SLIDERTRACK, with the slider set to the minimum position
CTRL+DOWN	SB_SLIDERTRACK, with the slider set to the maximum position
F7	SB_PAGEUP
F8	SB_PAGEDOWN

Using Scroll Bars

This section explains how to perform the following tasks:

- Create scroll bars
- Retrieve a scroll-bar handle
- Initialize, adjust, and read the scroll-bar range and position

Creating Scroll Bars

When creating a frame window, you can add scroll bars by specifying the FCF_HORZSCROLL flag, FCF_VERTSCROLL flag, or both flags in the WinCreateStdWindow function. This adds horizontal, vertical, or both (as specified) scroll bars to the frame window. The frame window owns the scroll bars and passes notification messages from the scroll bars to the client window. The following code fragment adds scroll bars to a frame window:

```
/* Set flags for a main window with scroll bars. */
ULONG ulFrameControlFlags =
    FCF_STANDARD | FCF_HORZSCROLL | FCF_VERTSCROLL;

/* Create the window. */
hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
    WS_VISIBLE,
    &ulFrameControlFlags,
    szClientClass,
    szFrameTitle,
    0,
    (HMODULE) NULL,
    0,
    &hwndClient);
```

Scroll bars created this way have the window identifier FID_HORZSCROLL or FID_VERTSCROLL. To determine the size and position of the scroll bars, the frame window uses the standard size specified by the system values SV_CXVSCROLL and SV_CYHSCROLL. The position always is defined by the right and bottom edges of the frame window.

Another way to create scroll bars is using the WinCreateWindow function. This method is most commonly used for stand-alone scroll bars. Creating scroll bars this way lets you set the size and position of the scroll bars. You also can specify which window should receive notification messages.

The following code fragment creates a stand-alone scroll bar:

```
#define ID_SCROLL_BAR 1

HWND hwndScroll, hwndClient;
hwndScroll = WinCreateWindow(
    hwndClient,                /* Scroll-bar parent window */
    WC_SCROLLBAR,              /* Preregistered scroll-bar class */
    (PSZ) NULL,                /* No window title */
    SBS_VERT | WS_VISIBLE,     /* Vertical style and visible */
    10, 10,                    /* Position & Size */
    20, 100,                   /* Size */
    hwndClient,                /* Owner */
    HWND_TOP,                  /* Z-order position */
    ID_SCROLL_BAR,             /* Scroll-bar identifier */
    NULL,                      /* No class-specific data */
    NULL);                     /* No presentation parameters */
```

Retrieving a Scroll-Bar Handle

If you use the `WinCreateStdWindow` function to create a scroll bar as a child of the frame window, you must be able to retrieve the scroll-bar handle. One way to do this is to use the `WinWindowFromID` function, the frame-window handle, and a predefined identifier (such as `FID_HORZSCROLL` or `FID_VERTSCROLL`), as shown in the following code fragment:

```
HWND hwndFrame,hwndHorzScroll,hwndVertScroll;

hwndHorzScroll = WinWindowFromID(hwndFrame, FID_HORZSCROLL);
hwndVertScroll = WinWindowFromID(hwndFrame, FID_VERTSCROLL);
```

If the standard frame window includes a client window, you can use that handle to access the scroll bars. The idea is to get the frame-window handle first; then, the scroll-bar handle.

```
HWND hwndScroll,hwndClient;

/* Get a handle to the horizontal scroll bar. */
hwndScroll = WinWindowFromID(
    WinQueryWindow(hwndClient, QW_PARENT),
    FID_HORZSCROLL);
```

Using the Scroll-Bar Range and Position

You can initialize the current value and range of a scroll bar to non-default values by sending the `SBCDATA` structure with class-specific data for a call to `WinCreateWindow`:

```
#define ID_SCROLL_BAR 1

SBCDATA sbcd;
HWND hwndScroll,hwndClient;

/* Set up scroll-bar control data. */
sbcd.posFirst = 200;
sbcd.posLast  = 400;
sbcd.posThumb = 300;

/* Create the scroll bar. */
hwndScroll = WinCreateWindow(hwndClient,
    WC_SCROLLBAR,
    (PSZ) NULL,
    SBS_VERT | WS_VISIBLE,
    10, 10,
    20, 100,
    hwndClient,
    HWND_TOP,
    ID_SCROLL_BAR,
    &sbcd, /* Class-specific data */
    NULL);
```

You can adjust a scroll-bar value and range by sending it an `SBM_SETSCROLLBAR` message:

```
/* Set the scroll-bar value and range. */

WinSendMsg(hwndScroll, SBM_SETSCROLLBAR,
    (MPARAM) 300,
    MPFROM2SHORT(200, 400));
```

You can read a scroll-bar value by sending it an `SBM_QUERYPOS` message:

```

USHORT usSliderPos;

/* Read the scroll-bar value. */
usSliderPos = (USHORT) WinSendMsg(hwndScroll,
    SBM_QUERYPOS, (MPARAM) NULL, (MPARAM) NULL);

```

Similarly, you can set a scroll-bar value by sending an SBM_SETPOS message:

```

/* Set the vertical scroll-bar value. */
WinSendMsg(hwndScroll, SBM_SETPOS, (MPARAM) 300, (MPARAM) NULL);

```

You can read a scroll-bar range by sending it an SBM_QUERYRANGE message:

```

MRESULT mr;
USHORT usMinimum, usMaximum;

/* Read the vertical scroll-bar range. */
mr = WinSendMsg(hwndScroll, SBM_QUERYRANGE, (MPARAM) NULL, (MPARAM) NULL);

usMinimum = SHORT1FROMMR(mr);          /* minimum in the low word */
usMaximum = SHORT2FROMMR(mr);          /* maximum in the high word */

```

Slider Controls

A slider control is a visual component that displays a range of values and allows a user to set, display, or modify a value by moving a slider arm. There are two types of sliders:

- The *linear slider* is represented as a shaft along which the slider arm can be moved by the user to set a value.
- The *circular slider* is represented as a dial with the slider arm shown as the radius of the dial.

This chapter explains how to use each of these slider controls in PM applications.

About Slider Controls

This section covers linear and circular slider controls. Linear sliders are used to set values that have familiar increments. Circular sliders, although different in appearance from linear sliders, provide much the same function. Both types of sliders can be used in a window to create a user interface.

Linear Sliders

Typically, linear sliders are used to easily set values that have familiar increments, such as feet, inches, degrees, and decibels. They also can be used for other purposes when immediate feedback is required, such as blending colors or showing a task's percentage of completion. For example, an application might let a user mix and match color shades by moving a slider arm, or a read-only slider could show how much of a task is complete by filling in the slider shaft as the task progresses. These are just a few examples of the ways in which sliders can be used.

The *slider arm* shows the value currently set by its position on the *slider shaft*. The user selects slider values by changing the location of the slider arm.

A *tick mark* indicates an incremental value in a slider scale. A *detent*, which is similar to a tick mark, also represents a value on the scale. However, a detent can be placed anywhere along the slider scale, rather than only in specific increments, and can be selected.

The appearance of a slider and the user interaction with a slider are similar to that of a scroll bar. However, these two controls are not interchangeable because each has a unique purpose. A scroll bar scrolls information into view that is outside a window's work area, while the slider is used to set, display, or modify that information, whether it is in or out of the work area.

Although linear sliders usually use values that have familiar increments, text also can be used. However, if the text is too long it can overlap the text displayed on the next tick mark or detent. Also, if the text on the far edge markers is too long, some of the text will not be displayed on screen. To prevent this use one of the following:

- Smaller font
- Shorter text values
- Static controls

The slider can be customized to meet varying application requirements, while providing a user interface component that can be used easily to develop products that conform to the Systems Application Architecture (SAA) Common User Access (CUA) user interface guidelines. The application can specify different scales, sizes, and orientations for its sliders, but the underlying function of the control remains the same. For a complete description of CUA sliders, refer to the *SAA CUA Guide to User Interface Design* and the *SAA CUA Advanced Interface Design Reference*.

Linear Slider Styles

Slider control styles are set when a slider window is created. The following table describes linear slider control styles. If no styles are specified, defaults are used as indicated in the table.

Style Name	Description
SLS_BOTTOM	Positions the slider at the bottom of the slider window. Valid only for horizontal sliders.
SLS_BUTTONSBOTTOM	Specifies that the optional slider buttons are to be used and places them at the bottom of the slider shaft. The buttons move the slider arm by one position, up or down, in the direction selected. Valid only for vertical sliders.
SLS_BUTTONSLEFT	Specifies that the optional slider buttons are to be used and places them to the left of the slider shaft. The buttons move the slider arm by one position, left or right, in the direction selected. Valid only for horizontal sliders.
SLS_BUTTONSRIGHT	Specifies that the optional slider buttons are to be used and places them to the right of the slider shaft. The buttons move the slider arm by one position, left or right, in the direction selected. Valid only for horizontal sliders.
SLS_BUTTONSTOP	Specifies that the optional slider buttons are to be used and places them at the top of the slider shaft. The buttons move the slider arm by one position, up or down, in the direction selected. Valid only for vertical sliders.
SLS_CENTER	Centers the slider within the slider window. This is the default position of the slider.

SLS_HOMEBOTTOM	Specifies the slider arm's home position. The bottom of the slider is used as the base value for incrementing. Valid only for vertical sliders.
SLS_HOMELEFT	Specifies the slider arm's home position. The left edge is used as the base value for incrementing. This is the default for horizontal sliders and is valid only for horizontal sliders.
SLS_HOMERIGHT	Specifies the slider arm's home position. The right edge is used as the base value for incrementing. Valid only for horizontal sliders.
SLS_HOMETOP	Specifies the slider arm's home position. The top of the slider is used as the base value for incrementing. Valid only for vertical sliders.
SLS_HORIZONTAL	Positions the slider horizontally. The slider arm can move left and right on the slider shaft. A scale can be placed on top of the slider shaft, below the slider shaft, or in both places. This is the default orientation of the slider.
SLS_LEFT	Positions the slider at the left edge of the slider window. Valid only for vertical sliders.
SLS_OWNERDRAW	Notifies the application whenever the slider shaft, the ribbon strip, the slider arm, and the slider background are to be drawn.
SLS_PRIMARYSCALE1	Determines the location of the scale on the slider shaft by using increment and spacing specified for scale 1 as the incremental value for positioning the slider arm. Scale 1 is displayed above the slider shaft of a horizontal slider and to the right of the slider shaft of a vertical slider. This is the default for a slider.
SLS_PRIMARYSCALE2	Determines the location of the scale on the slider shaft by using increment and spacing specified for scale 2 as the incremental value for positioning the slider arm. Scale 2 is displayed below the slider shaft of a horizontal slider and to the left of the slider shaft of a vertical slider.
SLS_READONLY	Creates a read-only slider, which presents information to the user but allows no interaction with the user.
SLS_RIBBONSTRIP	Fills, as the slider arm moves, the slider shaft between the home position and the slider arm with a color value different from slider shaft color, similar to mercury in a thermometer.
SLS_RIGHT	Positions the slider at the right edge of the slider window. Valid only for vertical sliders.
SLS_SNAPTOINCREMENT	Causes the slider arm, when positioned between two values, to be positioned to the nearest value and redrawn at that position.

SLS_TOP	Positions the slider at the top of the slider window. Valid only for horizontal sliders.
SLS_VERTICAL	Positions the slider vertically. The slider arm can move up and down the slider shaft. A scale can be placed on the left side of the slider shaft, on the right side of the slider shaft, or in both places.

More on Linear Slider Styles

This section summarizes information in the table and provides additional information on some of the styles.

Slider Orientation

The slider's orientation is determined by specifying SLS_HORIZONTAL or SLS_VERTICAL. The default orientation is horizontal, with the slider arm moving left and right on the shaft.

Slider Positioning

The slider's positioning within the slider window is determined by specifying SLS_CENTER, SLS_BOTTOM, SLS_TOP, SLS_LEFT, or SLS_RIGHT. The default positioning is centered in the slider window.

Slider Scale Location

The location of the scale on the slider shaft is determined by specifying SLS_PRIMARYSCALE1 or SLS_PRIMARYSCALE2. The default is to use the increment and spacing specified for scale 1 as the incremental value for positioning the slider arm. Scale 1 is displayed above the slider shaft of a horizontal slider and to the right of the slider shaft of a vertical slider.

Slider Arm Home Position

The slider arm's home position is determined by specifying SLS_HOMELEFT, SLS_HOMERIGHT, SLS_HOMEBOTTOM, or SLS_HOMETOP. The default is SLS_HOMELEFT for horizontal sliders and SLS_HOMEBOTTOM for vertical sliders.

Slider Buttons Location

The location of the slider buttons, if used, is determined by specifying SLS_BUTTONSLEFT, SLS_BUTTONSRIGHT, SLS_BUTTONSBOTTOM, or SLS_BUTTONSTOP. If you do not specify one of these styles, or if conflicting styles are specified, slider buttons are not included in the slider control.

Other Styles

If SLS_SNAPTOINCREMENT is specified and the slider arm is moved to a position between two values on the slider scale, it is positioned on the nearest value and redrawn at that position. If this style is not specified, the slider arm remains at the position to which it is moved.

SLS_READONLY creates a read-only slider. This means the user cannot interact with the slider. It is simply used as to present a quantity to the user, such as the percentage of completion of an ongoing task. Visual differences for a read-only slider include a narrow slider arm, no slider buttons, and no detents.

The SLS_RIBBONSTRIP style allows movement of the slider arm to cause the slider shaft between home position and the slider arm to be filled with a color value that is different from the slider shaft color, similar to mercury in a thermometer.

If SLS_OWNERDRAW is specified, the application is notified whenever the slider shaft, the ribbon strip, the slider arm, and the slider background are to be drawn.

Circular Sliders

The circular slider, although different in appearance from the linear slider, provides much the same function. The circular slider control provides an analog user interface and emulates the controls of stereo and video equipment. Because the circular slider takes up less space on the screen, it may be more practical to use in cases where you might want to have several controls in the same window. You may want to use both types of sliders in a window to create a user interface that makes good use of available space and provides a familiar appearance to the user.

The slider arm shows the value currently set by its position on the slider dial. The slider arm can also be represented as a small circular thumb on the dial rather than a line. The user selects slider values by changing the location of the slider arm on the dial. Outside the perimeter of the dial is a circular scale with tick marks representing incremental values the slider arm can point to. Its values can be tracked by pointing to any area on the dial and pressing the select button while moving the mouse on the desktop.

The circular slider can have a set of buttons, one to the left and the other to the right of the scroll range, similar to the buttons found on the linear slider, that can be used to modify the value shown on the slider.

The minus sign on the left button and plus sign on the right button can be replaced with other symbols. For example, you might want to use a left arrow and a right arrow instead of the minus and plus signs.

Another option of the circular slider is to have a window, in the center of the dial, that displays the value of the dial in scrollable numeric text.

The appearance and functionality of the circular slider are controlled by the circular slider control styles specified. These style bits are summarized in the section that follows.

Circular Slider Styles

Circular slider control style bits control the appearance and behavior of the slider. The following table describes circular slider control styles:

Style Name	Description
CSS_360	Extends the scroll range of the dial to 360 degrees. When this style is set, CSS_NONUMBER and CSS_NOBUTTON

	styles automatically are set.
CSS_CIRCULARVALUE	Draws a circular thumb, rather than a line, for the value indicator.
CSS_MIDPOINT	Enlarges the mid-point and end-point tick marks.
CSS_NOBUTTON	Prevents the display of the + and - value buttons.
CSS_NONUMBER	Prevents the display of a scrollable numeric value on the dial indicating the dial's value.
CSS_NOTEXT	Prevents the display of a window title beneath the dial.
CSS_POINTSELECT	Enables tracking of the dial's value with the mouse.
CSS_PROPORTIONALTICKS	Enables the length of the tick marks to be calculated as a percentage of the dial's radius.

More on Circular Slider Styles

This section provides information on some of the styles.

Circular Slider Buttons

The circular slider has a set of buttons, one to the left and the other to the right of the scroll range. These buttons are similar to the buttons found on the linear slider. When selected, they modify the value of the circular slider in opposing ways. If you decide you do not want to display these buttons as part of your circular slider, specify the style CSS_NOBUTTON.

The bit maps that show a minus sign (-) on the left button and a plus sign (+) on the right button can be replaced. For example, you might want to use a left arrow (←) and a right arrow (→). To set the bit map data for the replacement bit maps, the owner window must send the CSM_SETBITMAPDATA control message. The optimal size for the button bit maps is 10x10 pels.

Window Title

Centered beneath the dial of the circular slider is a rectangular text field that contains the title of the window. To prevent the display of this feature, specify CSS_NOTEXT.

Dial Value Window

Another option of the circular slider is a window, in the center of the dial, that displays the value of the dial in scrollable numeric text. To

prevent the display of this window, specify `CSS_NONUMBER`.

360-Degree Scale

You can choose a scale of 360 degrees for your slider with `CSS_360`. Setting this style causes the `CSS_NOBUTTON` and `CSS_NONUMBER` styles to be set automatically. The `CSS_NONUMBER` style prevents the value indicator from corrupting the dial value. A 360-degree circular slider is displayed without the bit maps for the plus and minus buttons.

Tracking Modes for Direct Manipulation

There are two tracking modes used for direct manipulation of the circular slider: scrolling the dial and point selection.

The default tracking behavior is scrolling, where the dial position of the slider is changed gradually. For example, by holding down the select mouse button while the pointer is positioned on the indicator line of the dial, you can move the mouse and cause the dial to rotate. This gradual changing of the dial value might be desirable for a volume control.

If the `CSS_POINTSELECT` style is specified for the circular slider, the position of the dial is changed immediately to a point on the scale that has been selected by the mouse. Point selection allows value changes to occur immediately.

Sizing Tick Marks

When a low-resolution display is being used, or in situations where the circular slider must be made very small, the length of the tick marks can be sized proportionately to allow effective sizing of the circular slider. Specifying the `CSS_PROPORTIONALTICKS` style causes the length of the tick marks to be calculated as a percentage of the dial's radius. This style does not adversely affect the size of the push buttons and bit-map graphics an application might provide.

Using Slider Controls

This section explains how to use sliders in your PM applications. It covers:

- Creating a linear slider
- Retrieving data for selected slider values
- Creating a circular slider

Code samples are provided.

Creating a Linear Slider

Before the slider is created, a temporary `SLDCDATA` data structure is allocated, and variables are specified for the slider control window handle and slider style. The `SLDCDATA` data structure is allocated so that the scale increments and spacing of the slider can be specified.

The slider style variable enables the application to specify style bits, `SLS_*` values, that are used to customize the slider.

You create a slider by using the `WC_SLIDER` window class name in the *ClassName* parameter of `WinCreateWindow` call. The handle of the slider control window is returned in the slider window variable.

After the slider is created, but before it is made visible, the application can set other slider control characteristics, such as:

- Size and placement of tick marks
- Text above one or more tick marks
- One or more detents
- Initial slider arm position

The settings in the preceding list are just a few that an application can specify. Slider control messages are used to specify these settings.

The sample code in the following figure shows an example of how a linear slider is created. The main components of the slider are labeled.

```
SLDCDATA sldcData;           /* SLDCDATA data structure */
CHAR      szTickText[5];     /* Text strings variable */
USHORT    idx;               /* Counter for setting text */
                                /* strings */
HWND      hwndSlider;        /* Slider window handle */
ULONG     ulSliderStyle;     /* Slider styles */

/*****
/* Initialize the parameters in the data structure.
*****/
sldcData.cbSize = sizeof(SLDCDATA); /* Size of SLDCDATA structure */
sldcData.usScale1Increments = 6;    /* Number of increments */
sldcData.usScale1Spacing = 0;       /* Use 0 to have slider calculate */
                                    /* spacing */

/*****
/* Set the SLS_* style flags to the default values, plus slider
/* buttons right.
*****/
ulSliderStyle = SLS_HORIZONTAL | /* Slider is horizontal */
                SLS_CENTER |     /* Slider shaft centered in */
                SLS_HOMELEFT |   /* Home position is left edge of */
                SLS_PRIMARYSCALE1 | /* Scale is displayed above */
                SLS_BUTTONSRIGHT; /* Slider buttons at right end of */

/*****
/* Create the slider control window.
/* The handle of the window is returned in hwndSlider.
*****/
hwndSlider = WinCreateWindow(
    hwndClient,           /* Parent window handle */
    WC_SLIDER,            /* Slider window class name */
    (PSZ)NULL,           /* No window text */
    ulSliderStyle,        /* Slider styles variable */
    (SHORT)10,            /* X coordinate */
    (SHORT)10,            /* Y coordinate */
    (SHORT)150,           /* Window width */
    (SHORT)80,            /* Window height */
    hwndClient,           /* Owner window handle */
    HWND_TOP,             /* Sibling window handle */
    ID_SLIDER,            /* Slider control window ID */
    &sldcData,            /* Control data structure */
    (PVOID)NULL);        /* No presentation parameters */

/*****
/* Set tick marks at several places on the slider shaft using the
/* primary scale.
*****/
WinSendMsg(hwndSlider,    /* Slider window handle */
    SLM_SETTICKSIZE,      /* Message for setting tick mark */
    /* size. */
    MPFROM2SHORT(
        SMA_SETALLTICKS, /* Attribute for setting all tick */
        6),              /* marks to the same size */
    /* Draw tick marks 6 pixels long */
    NULL);               /* Reserved value */

/*****/
```

```

/* Set text above the tick marks. */
/*****
for (idx = 0; idx <= 5; idx++) /* Count from 0 to 5 */
{
    itoa(10*idx, szTickText, 10); /* Set text at increments of 10 */

    WinSendMsg(hwndSlider, /* Slider window handle */
        SLM_SETSCALETEXT, /* Message for setting text on a */
        /* slider scale */
        MPFROMSHORT(idx), /* Text string counter */
        MPFROMP(szTickText)); /* Text to put on slider scale */
}

/*****
/* Set detents between two of the tick marks on the slider shaft. */
/*****
WinSendMsg(hwndSlider, /* Slider window handle */
    SLM_ADDDETENT, /* Message for adding detents to */
    /* a slider scale */
    MPFROMSHORT(5), /* Put a detent 5 pixels from home */
    NULL); /* Reserved value */

WinSendMsg(hwndSlider, /* Slider window handle */
    SLM_ADDDETENT, /* Message for adding detents to */
    /* slider scale */
    MPFROMSHORT(25), /* Put a detent 25 pixels from */
    /* home */
    NULL); /* Reserved value */

/*****
/* Set the slider arm position to the 1st increment on the scale. */
/*****
WinSendMsg(hwndSlider, /* Slider window handle */
    SLM_SETSLIDERINFO, /* Message for setting slider */
    /* attributes */
    MPFROM2SHORT(
        SMA_SLIDERARMPOSITION, /* Modify slider arm position */
        SMA_INCREMENTVALUE), /* Use an increment value */
    MPFROMSHORT(1)); /* Value to use is 1st */
    /* increment */

/*****
/* Since all items have been set, make the control visible. */
/*****
WinShowWindow(hwndSlider, /* Slider window handle */
    TRUE); /* Make the window visible */

```

Retrieving Data for Selected Slider Values

To retrieve data represented by a slider value, specify a variable for the current position of the slider arm. Then, use the `SLM_QUERYSLIDERINFO` message to retrieve information about the current slider arm position in increment coordinates. The code fragment in the following figure shows how to retrieve data for a selected slider value:

```

ULONG    ulValue; /* Variable in which to store */
           /* current slider arm position */

/*****
/* Get the information about the current slider arm position in */
/* incremental coordinates. */
/*****
ulValue = (ULONG)WinSendMsg(
    hwndSlider, /* Slider window handle */
    SLM_QUERYSLIDERINFO, /* Message for querying slider */
    /* attributes */
    MPFROM2SHORT(
        SMA_SLIDERARMPOSITION, /* Get increment at which slider */
        SMA_INCREMENTVALUE), /* arm is located */
    1);

```

```
NULL); /* Reserved value */
```

Creating a Circular Slider

The circular slider PM window class WC_CIRCULARSLIDER is similar to the window class of a linear slider or a scroll bar. This window class must be registered with WinRegisterCircularSlider before you can create a circular slider. A circular slider can be created by a CONTROL statement in a dialog resource, as shown in the following figure:

```
CONTROL "~Balance",
        ID_BALANCECS,
        10, 50, 60, 60,
        WC_CIRCULARSLIDER,
        WS_TABSTOP |
        WS_VISIBLE |
        CSS_POINTSELECT
```

A circular slider also can be created by specifying the WC_CIRCULARSLIDER window class name as a parameter of the WinCreateWindow call, as shown in the following sample code:

```
hwndCS = WinCreateWindow (hwndClient, /* Parent handle */
                          WC_CIRCULARSLIDER, /* Class name */
                          "~Balance", /* Window text */
                          WS_VISIBLE |
                          WS_TABSTOP |
                          CSS_POINTSELECT,
                          0,0,0,0, /* Coordinates */
                          hwndClient, /* Owner handle */
                          HWND_TOP, /* Z-order */
                          ID_BALANCECS, /* Window ID */
                          NULL, /* Control data */
                          NULL); /* Presparam */
```

Circular Slider Sample

The following is a complete coding example for adding a circular slider, and includes the following files:

- CIRCLE.C
- CIRCLE.RC
- CIRCLE.H

```
=====
CIRCLE.C
=====
#define INCL_WIN

#include <os2.h>
#include "circle.h"

/* Procedure Prototype */
MRESULT EXPENTRY MyWindowProc(HWND hwnd,ULONG msg,MPARAM mp1,MPARAM mp2);
MRESULT EXPENTRY MainProc(HWND hwnd,ULONG msg,MPARAM mp1,MPARAM mp2);

/* Global Variables */
```



```

HAB      hab;
HMQ      hmq;
QMSG     qmsg;
HWND     hwndFrame;
ULONG    flCreate;
HWND     hwndClient;

INT main(VOID)
{
    /* Convert system pointer into hourglass pointer */
    WinSetPointer(HWND_DESKTOP,
        WinQuerySysPointer(HWND_DESKTOP,SPTR_WAIT,FALSE));

    hab = WinInitialize(0);
    hmq = WinCreateMsgQueue(hab,0);

    WinRegisterClass(hab,"Client",MainProc,CS_SIZEREDRAW,0);

    flCreate = FCF_SYSMENU      |
               FCF_SIZEBORDER  |
               FCF_TITLEBAR    |
               FCF_MENU        |
               FCF_MINMAX      |
               FCF_SHELLPOSITION |
               FCF_TASKLIST;

    hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                   WS_VISIBLE,
                                   &flCreate,
                                   "Client",
                                   "My Dial",
                                   0L, 0,
                                   MAIN_FRAME,
                                   &hwndClient);

    /* Convert system pointer into arrow pointer */
    WinSetPointer(HWND_DESKTOP,
        WinQuerySysPointer(HWND_DESKTOP,SPTR_ARROW,FALSE));

    while (WinGetMsg(hab,&qmsg,0,0,0))WinDispatchMsg(hab,&qmsg);

    WinDestroyWindow(hwndFrame);
    WinDestroyMsgQueue(hmq);
    WinTerminate(hab);

    /* Beep when done */
    DosBeep(750,500);
    return(0);
}

MRESULT EXPENTRY MainProc(HWND hwnd,ULONG msg,MPARAM mp1,MPARAM mp2)
{
    HPS     hps;
    static  HWND hwndCirc;
    SWP     swp;
    switch(msg)
    {
        case WM_CLOSE:
            WinPostMsg(hwnd,WM_QUIT,0L,0L);
            return ((MRESULT)NULL);

        case WM_COMMAND:
            /* Exit option was selected in the menu bar */
            switch(SHORT1FROMMP(mp1))
            {
                case IDM_FILEEXIT:
                    WinPostMsg(hwnd,WM_QUIT,0L,0L);
                    return ((MRESULT)NULL);
            }
            return ((MRESULT)NULL);

        case WM_CONTROL:
            /* Process circular slider notification messages */
            if (SHORT1FROMMP(mp1) == ID_DIAL)
            {
                switch (SHORT2FROMMP(mp1))
                {
                    /* Notification codes can be specified here */

```

```

    }
}
/* Default processing for other control window ids */
return (WinDefWindowProc(hwnd,msg,mp1,mp2));

case WM_CREATE:
/* Create circular slider control */
hwndCirc = WinCreateWindow(hwnd,
    WC_CIRCULARSLIDER,
    "My Dial Window",
    WS_VISIBLE,
    0, 0, 0, 0,          /* Position & size */
    hwnd,               /* Client window */
    HWND_TOP,
    ID_DIAL,
    NULL,NULL);

/* Specify range of values for circular slider */
WinSendMsg (hwndCirc,
    CSM_SETRANGE,
    MPFROMLONG(0L),
    MPFROMLONG(100L));

/* Specify scroll & tick mark increments */
WinSendMsg (hwndCirc,
    CSM_SETINCREMENT,
    MPFROMLONG(10L),
    MPFROMLONG(2L));

/* Set initial value */
WinSendMsg (hwndCirc,
    CSM_SETVALUE,
    MPFROMLONG(80L),
    NULL);

return (MRESULT)FALSE;

case WM_SIZE:
/* The frame window has changed in size */
/* Recalculate size of circular slider */
WinQueryWindowPos(hwnd,&swp);
WinSetWindowPos(hwndCirc,
    HWND_TOP,
    0, 0,
    swp.cx,
    swp.cy,
    SWP_MOVE |
    SWP_SIZE);
return (MRESULT)NULL;

case WM_PAINT:
hps = WinBeginPaint(hwnd,0,NULL);
WinEndPaint(hps);
return (MRESULT)NULL;

default:
return (WinDefWindowProc(hwnd,msg,mp1,mp2));
}
}

=====
CIRCLE.RC
=====
#include <os2.h>
#include "circle.h"

ACCELTABLE MAIN_FRAME
{
    VK_F3, IDM_FILEEXIT, VIRTUALKEY
}

MENU    MAIN_FRAME
{
    SUBMENU    "~File",          IDM_FILEMENU
    {
        MENUITEM    "E~xit\tF3",    IDM_FILEEXIT
    }
}

```

```
=====
CIRCLE.H
=====
#define MAIN_FRAME      255
#define IDM_FILEMENU    256
#define IDM_FILEEXIT    257
#define ID_DIAL         258
```

Graphical User Interface Support for Slider Controls

This section describes the support the slider control provides for graphical user interfaces (GUIs). Except where noted, this support conforms to the guidelines in the *SAA CUA Advanced Interface Design Reference*.

Since slider values all are mutually exclusive, only one of them can be selected at a time. Therefore, the only type of selection supported by the slider control is *single selection*.

Note: If more than one slider window is open, selecting values in one slider window has no effect on the values selected in any other slider window. For linear sliders, a black square is drawn in the center of the slider arm to show which slider control window has the focus.

An initial value is selected when the slider control is first displayed. If the application does not provide the initial selection for a linear slider (using the SLM_SETSLIDERINFO message) to position the slider arm, the value at the home position is selected automatically. The home position is the end of the slider that contains the lowest value on the scale.

Slider Navigation Techniques

The slider control supports the use of pointing devices and the keyboard for selecting values.

Pointing Device Support

A user can select slider values with a pointing device. The CUA guidelines defines mouse button 1 (the select button) as the button for selecting values, and button 2 (the drag button) for dragging the slider arm to a value. These definitions also apply to the same buttons on any other pointing device, such as a joystick.

The select button and drag button can be used in conjunction with the following slider components to select slider values:

- Slider arm

Moving the pointer over the slider arm, then pressing and holding the select or drag buttons while moving the pointer, causes the slider arm to move in the direction the pointer is moving. When the button is released, the value closest to the slider arm position becomes the selected value.

- Slider shaft

Clicking the select button when the pointer is over the slider shaft causes the slider arm to move one increment in the direction of the pointer. For linear sliders, increments are determined by the initial values passed for the primary scale specified (SLS_PRIMARYSCALE1 or SLS_PRIMARYSCALE2) when the slider is created.

Clicking the drag button when the pointer is over the slider shaft causes the slider arm to move to the pointer's location.

- Slider buttons

Clicking the select button when the pointer is over a slider button causes the slider arm to move one increment in the direction the arrow on the slider button is pointing.

Slider buttons are optional. If used, two slider buttons are available to the user. The arrows on top of the slider buttons point to opposite ends of the slider. Both slider buttons are positioned at the same end of the slider.

For linear sliders, slider buttons are enabled by specifying the appropriate SLS_* value when the slider control window is created. For horizontal sliders, you can specify either SLS_BUTTONSLEFT or SLS_BUTTONSRIGHT. For vertical sliders, you can specify either SLS_BUTTONSBOTTOM or SLS_BUTTONSTOP. The default is no slider buttons. If more than one of these style bits is specified, no slider buttons are enabled.

- Detents

A detent is similar to a tick mark on a linear slider scale because it represents a value on the scale. However, unlike a tick mark, a detent can be placed anywhere along the slider scale instead of in specific increments.

A detent can be selected by moving the pointer over it and pressing the select button on the pointing device. When this happens, the slider arm moves to the position on the slider shaft indicated by the detent.

Keyboard Support

A user can select a value by using the navigation keys to move the slider arm to the value or by typing a value in an entry field, if one is provided by the application, to change the slider arm position. The following list describes these methods of selecting slider values:

- Values can be selected using the Up, Down, Left, and Right Arrow keys to move the slider arm one increment at a time. The Up and Down Arrow keys are enabled for vertical sliders, and the Right and Left Arrow keys are enabled for horizontal sliders. If no tick mark exists on the scale in the requested direction, the slider arm does not move.

If an Arrow key is pressed in conjunction with the Shift key, the slider arm moves to the next detent instead of the next tick mark. If no detent exists on the scale in the requested direction, the slider arm does not move.
- The Home and End keys can be used to select the lowest and highest values, respectively, in the scale. If the Ctrl key is pressed in combination with the Home or End keys, the result is the same as pressing only the Home or End keys.
- The application can provide an optional entry field for the slider control. The entry field is a separate control, but it can work in conjunction with the slider control.

If the application provides an entry field for the slider control window, it must be implemented as follows:

- The user must be allowed to type a value into the entry field.
- If the typed value is within the range of the slider scale, the slider arm moves to that value as soon as the value is typed.
- No other action, such as pressing the Enter key, is required.

Spin Button Controls

A *spin button* control (WC_SPINBUTTON window class) is a visual component that gives users quick access to a finite set of data by letting them select from a scrollable ring of choices. Because the user can see only one item at a time, a spin button should be used only with data that is intuitively related, such as a list of the months of the year, or an alphabetic list of cities or states. This chapter explains when and how to use spin buttons in PM applications.

About Spin Button Controls

A *spin button* consists of at least one spin field that is a single-line entry (SLE) field, and up and down arrows that are stacked on top of one another. These arrows are positioned to the right of the SLE field.

You can create multi-field spin buttons for those applications in which users must select more than one value. For example, in setting a date, the spin button control can provide individual fields for setting the month, day, and year. The first spin field in the spin button could contain a list of months; the second, a list of numbers; and the third, a list of years.

The application uses a multi-field spin button by creating one master component that contains a spin field and the spin arrows, and servant components that contain only spin fields. The spin buttons are created at component initialization. The servant components are passed a handle to the master component in a message. When a servant spin field has the focus, it is spun by the arrows in the master component.

The list of values in a spin button entry field can be an array of data or a list of consecutive integers, defined by an upper and a lower limit.

Using Spin Button Controls

This section describes how to create a spin button control.

Creating a Spin Button

A spin button is created as a public window class by using `WinCreateWindow`, with a class style of `WC_SPINBUTTON` and a window style of `WS_VISIBLE`. These are joined with any of the spin button style flags by using a logical OR (`|`). The spin button style flags let you specify:

- Character input restrictions (*none, numeric, read-only*)
- Presentation of the data in the spin field (*left-justified, right-justified, centered*)
- Presence or absence of a border around the spin field
- Spin speed
- Zero-padding of numeric spin fields

The placement and width of the spin button component are specified as parameters in `WinCreateWindow`.

The upper and lower limits of numeric fields, the value array pointer for arrays of strings, and the initial value in the spin field are all set by messages sent from the application to the component.

You can destroy the spin button component window using `WinDestroyWindow` when finished. The component handle that was returned when the spin button was created is the input parameter to `WinDestroyWindow`. The following sample code shows an example of how to create a spin button.

```

ULONG          ulSpinStyle;          /* Spin Button style */
HWND           hwndSpin;              /* Spin Button window handle */

/*****
/* Set the SPBS_* style flags.
*****/
ulSpinStyle = SPBS_MASTER             /* Spin button has its own
                                     /* buttons,
                                     /* and it only holds numbers
SPBS_NUMERICONLY |                   /*
SPBS_JUSTRIGHT |                     /* that are right justified,
SPBS_FASTSPIN;                       /* and it spins faster as
                                     /* the arrows are held down

/*****
/* Create the Spin Button control window.
/* The handle of the window is returned in hwndSpin.
*****/
hwndSpin = WinCreateWindow (
    hwndClient,          /* Parent window handle */
    WC_SPINBUTTON,       /* Spin Button window class name */
    (PSZ)NULL,           /* No window text */
    ulSpinStyle,         /* Spin Button styles variable */

    (LONG)10,            /* X coordinate */
    (LONG)10,            /* Y coordinate */
    (LONG)150,           /* Window width */
    (LONG)50,            /* Window height */
    hwndClient,          /* Owner window handle */

```

```

        HWND_TOP,          /* Sibling window handle */
        ID_SPINBUTTON,     /* Spin Button control window ID */
        (PVOID)NULL,       /* No control data structure */
        (PVOID)NULL);      /* No presentation parameters */

/*****
/* Set the limits of the Spin Button control, since it has a style
/* of SPBS_NUMERICONLY.
*****/
WinSendMsg (hwndSpin,      /* Spin Button window handle */
            SPBM_SETLIMITS, /* Set limits message */
            (MPARAM)1000,   /* Spin Button maximum setting */
            (MPARAM)0);     /* Spin Button minimum setting */

/*****
/* Set the initial value of the Spin Button.
*****/
WinSendMsg (hwndSpin,      /* Spin Button window handle */
            SPBM_SETCURRENTVALUE, /* Set current value message */
            (MPARAM)100,     /* Spin Button initial value */
            (MPARAM)NULL);   /* Reserved value */

/*****
/* Because all items have been set, make the control visible.
*****/
WinShowWindow (hwndSpin,   /* Spin Button window handle */
               TRUE);       /* Make the window visible */

```

Graphical User Interface Support for Spin Button Controls

Users can interact with the spin button using either the keyboard or a pointing device, such as a mouse, as follows:

- Using the select button (button 1) on the pointing device, users first give focus to the spin field they want to change, and then click on either the Up Arrow or Down Arrow until the value they want is displayed in the spin field.
- Using a keyboard, users press the:
 - Up Arrow and Down Arrow keys to see the choices
 - Left Arrow and Right Arrow keys to move the cursor left and right within a spin field
 - Home and End keys to move the cursor to the first and last characters in a spin field
 - Tab and Shift+Tab keys to move the input focus from one field to another in multi-field spin buttons

Users can view the values in a spin field one at a time, or they can rapidly scroll a list by keeping either the Up or Down Arrow keys pressed. When a spin button is not read-only, users can advance quickly to the value they want to set in a spin field by typing over the value currently displayed.

Static Controls

A *static* control is a simple text field, bit map, or icon that an application can use to label, enclose, or separate other control windows. This chapter describes how to create and use static controls in PM applications.

About Static Controls

Unlike the other types of control windows, a static control does not accept user input or send notification messages to its owner. The primary advantage of a static control is that it provides a label or graphic that requires little attention from an application. At most, an application might change the text or position of a static control.

Keyboard Focus

A static control never accepts the keyboard focus. When a static control receives a WM_SETFOCUS message, or when a user clicks the static control, the system advances the focus to the next sibling window that is not a static control. If the control has no siblings, the system gives the focus to the owner of the static control.

Static Control Handle

Every static control is associated with a 32-bit data field. A static control with the SS_BITMAP or SS_ICON style uses this field to store the handle of the bit map or icon that it displays. An application can obtain that handle by sending the SM_QUERYHANDLE message to the control. An application can replace the bit map or icon by sending the SM_SETHANDLE message to the control, specifying a valid icon or bit map handle. Changing the handle causes the system to redraw the control.

For a non-icon or non-bit map static control, the data field is available for application-defined data and has no effect on the appearance of the control.

An application can retrieve the data field of a static control window by calling WinWindowFromID, using the handle of the owner and the window identifier of the static control. The static control window identifier is specified in either the dialog-window template or WinCreateWindow.

Static Control Styles

A static control has style bits that determine whether the control displays text, draws a simple box containing text, displays an icon or a bit map, or shows a framed or unframed colored box. Applications can specify a combination of the following styles for a static control:

Style Name	Description
SS_BITMAP	Draws a bit map. The bit map resource must be provided in the resource-definition file. To include the bit map in a dialog window, the resource identifier must be specified in the <i>text</i> parameter of the CONTROL statement in the resource definition file. To include the bit map in a non-dialog window, the ASCII representation of the identifier must be specified in the <i>pszName</i> parameter of WinCreateWindow, that is, the first byte of the <i>pszName</i> parameter must be the cross-hatch character (#), and the remaining text must be the ASCII representation of the identifier, for example, #125.
SS_BKGNDFRAME	Creates a box whose frame has the background color.
SS_BKGNDRECT	Creates a rectangle filled with the background color.
SS_FGNDFRAME	Creates a box whose frame has the

	foreground color.
SS_FGNDRECT	Creates a rectangle filled with the foreground color.
SS_GROUPBOX	Creates a box whose upper-right corner contains control text. This style is useful for enclosing groups of radio buttons or check boxes in a box.
SS_HALFTONEFRAME	Creates a box whose frame has halftone shading.
SS_HALFTONERECT	Creates a box filled with halftone shading.
SS_ICON	Draws an icon. The resource identifier for the icon resource is determined the same way as the SS_BITMAP style. The icon resource must be in the resource-definition file.
SS_SYSICON	Draws a system-pointer icon. The resource identifier for the system-pointer resource is determined the same way as the SS_BITMAP style. To display this system pointer, the system calls WinQuerySysPointer with the specified identifier.
SS_TEXT	Creates a box with formatted text. An application can combine various formatting options with this style to produce formatted text in the boundaries of the control. The formatting flags are the same as those used for WinDrawText.

Default Static Control Performance

The messages specifically handled by the predefined static control class (WC_STATIC) are as follows:

Message Name	Description
SM_QUERYHANDLE	Returns the handle associated with the static control window.
SM_SETHANDLE	Sets the handle associated with the static control and invalidates the control window, forcing it to be redrawn.
WM_ADJUSTWINDOWPOS	Adjusts the SWP data structure so that the new window size matches the bit map, icon, or system-pointer dimensions associated with the static control.
WM_CREATE	Sets the text for a static-text control. Loads the bit map or icon resource for the bit map or icon static control. Returns TRUE if the resource cannot be loaded.
WM_DESTROY	Frees the text for a static-text control. Destroys the bit map or icon for a bit map or icon static control. The icon for a system-pointer static

	control is not destroyed because it belongs to the system.
WM_ENABLE	Invalidates the entire static control window, forcing it to be redrawn.
WM_HITTEST	<p>Returns the value HT_TRANSPARENT for the following static-control styles:</p> <ul style="list-style-type: none"> o SS_BKGNDFRAME o SS_BKGNDRECT o SS_FGNDFRAME o SS_FGNDRECT o SS_GROUPBOX o SS_HALFTONEFRAME o SS_HALFTONERECT. <p>For other styles, this message returns the result of WinDefWindowProc.</p>
WM_MATCHMNEMONIC	Returns TRUE if the mnemonic passed in the <i>mpl</i> parameter matches the mnemonic in the control-window text.
WM_MOUSEMOVE	Sets the mouse pointer to the arrow pointer and returns TRUE.
WM_PAINT	Draws the static control based on its style attributes.
WM_QUERYDLGCODE	Returns the predefined constant DLGC_STATIC.
WM_QUERYWINDOWPARAMS	Returns the requested window parameters.
WM_SETFOCUS	Sets the focus to the next sibling window that can accept the focus; or if no such sibling exists, sets the focus to the parent window.
WM_SETWINDOWPARAMS	Allows the text to be set (static-text controls only).

Using Static Controls

This section explains how to perform the following tasks:

- Include a static control in a dialog window
- Include a static control in a client window

Including a Static Control in a Dialog Window

To include a static control in a dialog window, you must define the control in a dialog-window template in a resource-definition file. The following resource-definition file creates a dialog window that contains a static-text control and three static-icon controls:

```
DLGTEMPLATE IDD_TOOLDLG LOADONCALL MOVEABLE DISCARDABLE
BEGIN
    DIALOG "",
        IDD_TOOLDLG,
        114, 53, 161, 127,
```

```

        FS_NOBYTEALIGN |
        FS_DLGBOARDER |
        WS_VISIBLE |
        WS_SAVEBITS

BEGIN
    CTEXT "Select a tool",
        IDS_TEXT,
        49, 110, 56, 8,
        SS_TEXT |
        DT_CENTER |
        DT_TOP |
        WS_GROUP |
        WS_VISIBLE

    AUTORADIOBUTTON "Paintbrush",
        IDB_BRUSH,
        63, 87, 61, 10,
        WS_TABSTOP |
        WS_GROUP |
        WS_VISIBLE

    AUTORADIOBUTTON "Scissors",
        IDB_SCISSORS,
        63, 64, 60, 10,
        WS_TABSTOP |
        WS_VISIBLE

    AUTORADIOBUTTON "Eraser",
        IDB_ERASER,
        65, 39, 43, 10,
        WS_TABSTOP |
        WS_VISIBLE

    ICON IDI_BRUSH,
        IDI_BRUSHICON,
        33, 84, 22, 16,
        WS_GROUP |
        WS_VISIBLE

    ICON IDI_SCISSORS,
        IDI_SCISSORSICON,
        33, 60, 22, 16,
        WS_GROUP |
        WS_VISIBLE

    ICON IDI_ERASER,
        IDI_ERASERICON,
        33, 36, 22, 16,
        WS_GROUP |
        WS_VISIBLE

    PUSHBUTTON "OK",
        DID_OK,
        10, 12, 38, 13,
        WS_TABSTOP |
        WS_GROUP |
        WS_VISIBLE

    PUSHBUTTON "Cancel",
        DID_CANCEL,
        59, 12, 38, 13,
        BS_DEFAULT |
        WS_TABSTOP |
        WS_GROUP |
        WS_VISIBLE

    PUSHBUTTON "Help",
        IDB_HELP,
        111, 13, 38, 13,
        BS_HELP |
        WS_TABSTOP |
        WS_GROUP |
        WS_VISIBLE

END
END

ICON IDI_BRUSH    brush.ico
ICON IDI_SCISSORS scissr.ico

```

ICON IDI_ERASER eraser.ico

Including a Static Control in a Client Window

An application can include a static control in a non-dialog window by calling `WinCreateWindow` with the window class `WC_STATIC`. The *//Style* parameter to `WinCreateWindow` defines the appearance of the control.

The following code fragment creates a static text control whose size and position are based on the size of the client window and the metrics for the current font:

```
#define ID_TITLE 5
#define INCL_GPILCIDS

HWND hwnd,hwndStatic,hwndClient;
HPS hps;
RECTL rcl;
FONTMETRICS fm;
ULONG ulTitleLen;
CHAR szTitle[] = "Static Text Controls";

/* Obtain the size of the client window */
WinQueryWindowRect(hwnd, &rcl);

/* Obtain a presentation space handle and */
/* the metrics for the current font */
hps = WinBeginPaint(hwnd, (HPS) NULL, (PRECTL) NULL);
GpiQueryFontMetrics(hps, sizeof(FONTMETRICS), &fm);

/* Obtain the size of the static control text string */
ulTitleLen = (ULONG) strlen(szTitle);

/* Create the static control. Base the size and */
/* position on the size of the client window and */
/* the metrics of the current font. */

hwndStatic = WinCreateWindow(
    hwndClient,                /* Parent window */
    WC_STATIC,                 /* Window class */
    szTitle,                   /* Window text */
    WS_VISIBLE |               /* Make it visible */
    SS_TEXT |                  /* Static-text control */
    DT_VCENTER |               /* Center text vert. */
    DT_CENTER,                 /* Center text horiz. */

    ((rcl.xRight / 2) -        /* x position */
    (ulTitleLen / 2) * fm.lEmInc), /* y position */
    rcl.yTop - fm.lEmHeight * 2, /* Width */
    fm.lEmInc * ulTitleLen,      /* Height */
    hwndClient,                 /* Owner window */
    HWND_TOP,                   /* Top of z-order */
    ID_TITLE,                   /* Window identifier */
    NULL,                       /* Control data */
    NULL);                      /* Presentation parameters*/

WinEndPaint(hps);
```

If your application creates a static control with the `SS_ICON` or `SS_BITMAP` style, make sure that the resource identifier specified in the *pszName* parameter corresponds to an icon or a bit map resource in the resource-definition file. If there is no resource, the application cannot create the static control.

Title-Bar Controls

A *title bar* is one of several control windows that comprise a standard frame window, giving the frame window its distinctive look and performance capabilities. This chapter describes how to create and use title-bar control windows in PM applications.

About Title Bars

The title bar in a standard frame window performs the following four functions:

- Displays the title of the window across the top of the frame window
- Changes its highlighted appearance to show whether the frame window is active (Ordinarily, the topmost window on the screen is the active window.)
- Responds to the actions of the user-for example, dragging the frame window to a new location on the screen.
- Flashes (as a result of the WinFlashWindow function) to get the attention of the user.

Once the frame controls are in place in the frame window, an application typically ignores them, because the system handles frame controls. In some cases, however, an application can take control of the title bar by sending messages to the title-bar control window.

Default Title-Bar Behavior

A title-bar control window sends messages to its owner (the frame window) when the control receives user input. Following are the messages that the title-bar control processes. Each message is described in terms of how the title-bar control responds to that message.

Message	Description
TBM_QUERYHILITE	Returns the highlighted state of the title bar.
TBM_SETHILITE	Sets the highlighted state of the title bar, repainting the title bar if the state is changing.
WM_BUTTON1DBLCLK	Restores the title bar if the owner window is minimized or maximized. If the window is neither minimized nor maximized, this message maximizes the window.
WM_BUTTON1DOWN	Sends the WM_TRACKFRAME message to the owner window to start the tracking operation for the frame window.
WM_CREATE	Sets the text for the title bar. Returns FALSE if the text is already set.
WM_DESTROY	Frees the window text for the title bar.
WM_HITTEST	Always returns HT_NORMAL, so that the title bar does not beep when it is disabled. (It is disabled when the frame window is maximized.)
WM_PAINT	Draws the title bar.
WM_QUERYDLGCODE	Returns the predefined constant DLGC_STATIC. The user cannot use the Tab key to move to the title bar in a dialog

window.

`WM_QUERYWINDOWPARAMS` Returns the requested window parameters.

`WM_SETWINDOWPARAMS` Sets the specified window parameters.

`WM_WINDOWPOSCHANGED` Returns `FALSE`. Processes this message to prevent the `WinDefWindowProc` function from sending the size and show messages.

Using Title-Bar Controls

This section explains how to:

- Include a title bar in a frame window
- Alter the dragging action of a title bar

Including a Title Bar in a Frame Window

An application can include a title bar in a standard frame window by specifying the `FCF_TITLEBAR` flag in the `WinCreateStdWindow` function.

The following code fragment shows how to create a standard frame window with a title bar, minimize and maximize (window-sizing) buttons, size border, system menu, and an application menu.

```
#define ID_MENU_RESOURCE 101

HWND hwndFrame,hwndClient;
UCHAR szClassName[255];

ULONG flControlStyle = FCF_TITLEBAR | FCF_MINMAX | FCF_SIZEBORDER |
                      FCF_SYSTEMMENU | FCF_MENU;

hwndFrame = WinCreateStdWindow(HWND_DESKTOP, WS_VISIBLE | FS_ACCELTABLE,
                              &flControlStyle, szClassName, "",
                              0, (HMODULE) NULL, ID_MENU_RESOURCE,
                              &hwndClient);
```

To get the window handle of a title-bar control, an application calls `WinWindowFromID`, specifying the frame-window handle and a constant identifying the title-bar control, as shown in the following code fragment:

```
hwndTitleBar = WinWindowFromID(hwndFrame, FID_TITLEBAR);
```

To set the text of a title bar, an application can use the `WinSetWindowText` function. The frame window passes the new text to the title-bar control in a `WM_SETWINDOWPARAMS` message.

Altering Dragging Action

When the user clicks the title bar, the title-bar control sends a WM_TRACKFRAME message to its owner (the frame window). When the frame window receives the WM_TRACKFRAME message, the frame sends a WM_QUERYTRACKINFO message to itself to fill in a TRACKINFO structure that defines the tracking parameters and boundaries. To modify the default behavior, an application must subclass the frame window, intercept the WM_QUERYTRACKINFO message, and modify the TRACKINFO structure. If the application returns TRUE for the WM_QUERYTRACKINFO message, the tracking operation proceeds according to the information in the TRACKINFO structure. If the application returns FALSE, no tracking occurs.

Value Set Controls

A *value set control* (WC_VALUESET window class), like a radio button, is a visual component that enables a user to select one choice from a group of mutually exclusive choices. However, unlike radio buttons, a value set can use graphic images (bit maps or icons), as well as colors, text, and numbers, to represent the items a user can select. This chapter presents the basics about value set controls and tells you how to create and use them in PM applications.

About Value Set Controls

Even though text is supported, the purpose of a value set control is to display choices as graphic images for faster selection. The user can *see* the selections instead of having to take time to read descriptions of the choices. Using graphic images in a value set also lets you conserve space on the display screen. For example, if you want to let a user choose from a variety of patterns, you can present those patterns as value set choices. If long strings of data are to be displayed as choices, radio buttons should be used. However, for small sets of numeric or textual information, you can use either a value set or radio buttons.

The value set is customizable to meet varying application requirements, while providing a user interface component that can be used easily to develop products that conform to the Common User Access (CUA) user interface guidelines. The application can specify different types of items, sizes, and orientations for its value sets, but the underlying function of the control remains the same. For a complete description of CUA value sets, refer to the *SAA CUA Guide to User Interface Design* and the *SAA CUA Advanced Interface Design Reference*.

Value Set Styles

Value set control window styles are set when a value set window is created.

- Set one of the following styles when creating a value set control window. You can override these styles by specifying VIA_BITMAP, VIA_ICON, VIA_TEXT, VIA_RGB, or VIA_COLORINDEX attributes for individual value set items.

VS_BITMAP	The attribute for each value set item is set to the VIA_BITMAP value set item attribute, which means the value set treats each item as a bit map unless otherwise specified. This is the default.
VS_COLORINDEX	The attribute for each value set item is set to the VIA_COLORINDEX value set item attribute, which means the value set treats each item as an index into the logical color table unless otherwise specified. This style is most often used when the colors currently available are adequate.
VS_ICON	The attribute for each value set item is set to the VIA_ICON value set item attribute, which means the value set treats each item as an icon unless otherwise specified.
VS_RGB	The attribute for each value set item is set to the VIA_RGB value set item attribute, which means the value set treats each item as a RGB color value unless otherwise specified. This style is most often used when you need to create new colors.
VS_TEXT	The attribute for each value set item is set to the VIA_TEXT value set item attribute, which means the value set treats each item as a text string unless otherwise specified.

- Specify one or more of the following optional window styles, if desired, by using an OR operator (|) to combine them with the style specified from the preceding list:

VS_BORDER	The value set draws a thin border around itself to delineate the control.
VS_ITEMBORDER	The value set draws a thin border around each item to delineate it from other items. Note: The VS_ITEMBORDER style is useful for items that are hard to see, such as faint colors or patterns.
VS_OWNERDRAW	The application is notified whenever the background of the value set window is to be painted.
VS_RIGHTTOLEFT	The value set interprets column orientation as right-to-left, instead of the default left-to-right arrangement. This means columns are numbered from right-to-left with the rightmost column being 1 and counting up as you move left. Home is the rightmost column and end is the leftmost column. There is no visible difference between a value set ordered left-to-right and a value set ordered right-to-left. Therefore, if your application uses multiple value sets, the ordering of the items should be consistent in each value set to avoid confusing the user. Note: The VS_RIGHTTOLEFT style is used on creation of the control. Changing this style after creation causes unexpected results.
VS_SCALEBITMAPS	The value set automatically scales bit maps to the size of the cell. If this style is not used, each bit map is centered in its cell. Also, if the cell is smaller than the bit map, the bit map is clipped to the size of the cell.

Using Value Set Controls

This section provides information that will enable you to create and use a value set control effectively.

Creating a Value Set

You create a value set by using the WC_VALUESET window class name in the *ClassName* parameter of WinCreateWindow call.

Before the value set is created, a temporary *VSCDATA* data structure is allocated so that the number of rows and columns of the value set can be specified.

Also, VS_* values are specified in the *ulValueSetStyle* variable so that the value set can be customized. The following sample code shows the creation of a value set:

```
VSCDATA vscData;                /* VSCDATA data structure */
HWND     hwndValueSet;          /* Value set window handle */
ULONG     ulValueSetStyle;      /* Value set style variable */

/*****
/* Initialize the parameters in the data structure.
*****/
vscData.cbSize =                /* Size of value set equals size */
    sizeof(VSCDATA);           /* of VSCDATA */
vscData.usRowCount = 1;         /* 1 row in the value set */
vscData.usColumnCount = 3;      /* 3 columns in the value set */

/*****/
```

```

/* Set the VS_* style flags to customize the value set. */
/*****
ulValueSetStyle =
    VS_RGB | /* Use colors for items. */
    VS_ITEMBORDER | /* Put border around each value */
/* set item. */
    VS_BORDER; /* Put border around the entire */
/* value set */

/*****
/* Create the value set control window. */
/* The handle of the window is returned in hwndValueSet. */
/*****
hwndValueSet = WinCreateWindow(
    hwndClient, /* Parent window handle */
    WC_VALUESET, /* Value set class name */
    (PSZ)NULL, /* No window text */
    ulValueSetStyle, /* Value set styles */
    (SHORT)10, /* X coordinate */
    (SHORT)10, /* Y coordinate */

    (SHORT)300, /* Window width */
    (SHORT)200, /* Window height */
    hwndClient, /* Owner window handle */
    HWND_TOP, /* Z-order position */
    ID_VALUESET, /* Value set window ID */
    &vscData, /* Control data structure */
    (PVOID)NULL); /* No presentation parameters */

/*****
/* Set the color value for each item in each row and column. */
/*****
WinSendMsg(hwndValueSet, /* Value set window handle */
    VM_SETITEM, /* Message for setting items */
    MPFROM2SHORT(1,1), /* Set item in row 1, column 1 */
    MPFROMLONG(0x00FF0000)); /* to the color red. */

WinSendMsg(hwndValueSet, /* Value set window handle */
    VM_SETITEM, /* Message for setting items */
    MPFROM2SHORT(1,2), /* Set item in row 1, column 2 */
    MPFROMLONG(0x0000FF00)); /* to the color green. */

WinSendMsg(hwndValueSet, /* Value set window handle */
    VM_SETITEM, /* Message for setting items */
    MPFROM2SHORT(1,3), /* Set item in row 1, column 3 */
    MPFROMLONG(0x000000FF)); /* to the color blue. */

/*****
/* Set the default selection. */
/*****
WinSendMsg(hwndValueSet, /* Value set window handle */
    VM_SELECTITEM, /* Message for selecting items */
    MPFROM2SHORT(1,2), /* Item in row 1, column 2 */
    NULL); /* Reserved value */

/*****
/* Since all items have been set in the control, */
/* make the control visible. */
/*****
WinShowWindow(hwndValueSet, /* Value set window handle */
    TRUE); /* Make the window visible */

```

Retrieving Data for Selected Value Set Items

The next step is to be able to retrieve the data represented by a value set item. To do this, variables are specified for combined row and column index values, item attributes, and item information. Then the VM_QUERYSELECTEDITEM, VM_QUERYITEMATTR, and VM_QUERYITEM messages are used to retrieve the index values, attributes, and data. The following sample code shows how data for selected value set items is retrieved:


```

ULONG    ulIdx;                                /* Combined row and column */
                                                /* index value */
USHORT   usItemAttr;                           /* Item attributes */
ULONG    ulItemData;                           /* Item data */

/*****
/* Get the row and column index values of the item selected by the
/* user. These values are returned in the ulIdx parameter.
*****/
ulIdx = (ULONG)WinSendMsg(
    hwndValueSet,                                /* Value set window handle */
    VM_QUERYSELECTEDITEM,                       /* Message for querying
                                                /* the selected item */
    NULL, NULL);                                /* Reserved values */

/*****
/* Determine the type of item that was selected. This message is
/* only to determine how to interpret item data when a value set
/* contains different types of items.
*****/
usItemAttr = (USHORT)WinSendMsg(
    hwndValueSet,                                /* Value set window handle */
    VM_QUERYITEMATTR,                           /* Message for querying item attribute */
    MPFROMLONG(ulIdx),                          /* Row and column of selected item */
    NULL);                                       /* Reserved value */

/*****
/* Get the information about the selected (non-textual) item.
/* If you are dealing with text, you need to allocate a buffer
/* for the text string.
*****/
ulItemData = (ULONG)WinSendMsg(
    hwndValueSet,                                /* Value set window handle */
    VM_QUERYITEM,                               /* Message for querying an item */
    MPFROMLONG(ulIdx),                          /* Row and column of selected item */
    NULL);                                       /* Set to NULL because the item is not
                                                /* a text item */

```

Arranging Value Set Items

The application defines the arrangement of value set items; they can be arranged in one or more rows, columns, or both. Items are placed from left to right in rows and from top to bottom in columns. The application can change the number of rows and columns at any time.

The number of items that can be displayed depends on the number of items that fit into the spaces provided by the defined rows and columns. If the number of items exceeds the number of spaces, the excess items are not displayed.

You can change the composition of a value set by specifying new items. The new items either can be added to the value set or can replace existing items.

Graphical User Interface Support for Value Set Controls

This section describes the support the value set control provides for graphical user interfaces (GUIs). Except where noted, this support conforms to the guidelines in the *SAA CUA Advanced Interface Design Reference*.

The GUI support provided by the value set control consists of Navigating to and selecting value set items.

Value Set Navigation Techniques

Since all value set items are mutually exclusive, only one of them can be selected at a time. Therefore, the only type of selection supported by the value set control is *single selection*.

Note: If more than one value set window is open, navigating to and selecting items in one value set window has no affect on the items displayed in any other value set window.

An initial choice is selected when the value set control is first displayed. If the application does not provide the initial selection by using the VM_SELECTITEM message, the choice in row 1, column 1 is selected automatically.

The value set control supports the use of a pointing device, such as a mouse, and the keyboard for navigating to and selecting items, except for items that are dimmed on the screen. This dimming of items is called *unavailable-state emphasis* and indicates that the items cannot be selected. However, the *selection cursor*, a dotted outline that usually indicates that an item can be selected, can be moved to unavailable items so that a user can press F1 to determine why they cannot be selected. The following sections describe the pointing device and keyboard support for the value set control.

Pointing Device Support

A user can use a pointing device to select value set items. The *SAA CUA Guide to User Interface Design* defines mouse button 1, the *select* button, to be used for selecting items. This definition also applies to the same button on any other pointing device.

An item can be selected by moving the pointer of the pointing device to the item and clicking the select button. When this happens, a black box is drawn around the item to show that it has been selected. The black box is called *selected-state emphasis*. In addition, the selection cursor is drawn inside the black box.

Keyboard Support

The value set control supports *automatic selection*, which means that an available item is selected when the selection cursor is moved to that item. The item is given selected-state emphasis as soon as the selection cursor is moved to it. No further action, such as pressing the spacebar, is required. The same black box and dotted outline are used, for selected-state emphasis and the selection cursor respectively, as when an item is selected with a pointing device.

A user can navigate to and select an item by using either the navigation keys or mnemonic selection to move the selection cursor to the item, as described in the following list:

- Items can be selected using the Up, Down, Left, and Right Arrow keys to move the selection cursor from one item to another.
- The Home and End keys can be used to select the leftmost and rightmost items, respectively, in the current row. If the Ctrl key is pressed in combination with the Home or End key, the item in the top row and the leftmost column, or the item in the bottom row and the rightmost column, respectively, is selected.

Note: The preceding description assumes that the current style of the value set window is left-to-right. However, if the VS_RIGHTTOLEFT style bit is set, the directions described for the Home, End, Ctrl+Home, and Ctrl+End keys in the preceding paragraph are reversed.
- The PgUp key can be used to select the item in the top row that is directly above the current position of the selection cursor. The PgDn key can be used to select the item in the bottom row that is directly below the current position of the selection cursor. If the space in the top or bottom row directly above or below the current cursor position is blank, the cursor moves to the blank space.
- Another keyboard method of selecting items is *mnemonic selection*. A user performs mnemonic selection by pressing a character key that corresponds to an underlined character. Coding a tilde (~) before a text character in the item causes that character to be underlined and activates it as a mnemonic selection character. When this happens, the selection cursor is moved to the item that contains the underlined character, and that item is selected.

Enhancing Value Set Controls Performance and Effectiveness

This section provides dynamic resizing and scrolling to enable you to fine-tune a value set control.

Dynamic Resizing and Scrolling

The value set control supports *dynamic resizing* if the application sends the WM_SIZE message to a value set window. This means that the value set control automatically recalculates the size of the items when either the user or the application changes the size of the value set window.

If the value set window's size is decreased so that the window is not large enough to display all of the items the value set contains, the items are clipped. If scroll bars are desired to allow the clipped information to be scrolled into view, they must be provided by the application.

Windows

To most users, a *window* is a rectangular area of the display screen where an application receives input from the user and displays output. This chapter describes the parts of the operating system that enable a Presentation Manager (PM) application to create and use windows; manage relationships between windows; and size, move, and display windows. An overview of the following topics is presented:

- Window types, classes, and styles
- Window-creation techniques
- Window messages and message queues
- Methods of window input and output
- Window resources and procedures
- Window identification and modification

Subsequent chapters present more in-depth descriptions of windows, their advantages and uses, along with example code fragments.

About Windows

A PM application can interact with the user and perform tasks only by way of windows. Each window shares the screen with other windows, including those from other applications. The user employs the mouse and keyboard to interact with windows and their owner applications.

Desktop Window and Desktop-Object Window

The OS/2 operating system automatically creates the *desktop window* (known as the *workplace* in user terminology) when it starts a PM session.

```
Desktop Window
Main Window 3
    Main Window 2
        Main Window 1
            Chi
            Win
            2A
                Child Window 1a
                    Child Window 1b
```

Desktop Window Containing Windows of Several Applications The desktop window paints the background color of the screen and serves as the "progenitor" of all the windows displayed by all PM applications (but not of object windows, which do not require screen display). To make the desktop the parent in the WinCreateStdWindow function, you specify HWND_DESKTOP.

The windows immediately below the desktop are called *main* or *top-level* windows; these are called *primary windows* in user terminology. Every PM application creates at least one window to serve as the main window for that application. Most applications also create many other windows, directly or indirectly, to perform tasks related to the main window.

Each window helps display output and receive input from the user. The previous figure shows the desktop window containing windows of several applications. Notice that the main windows can overlap one another. (At times, it is possible for a main window to be completely hidden.) Operations in one main window normally do not affect the other main windows.

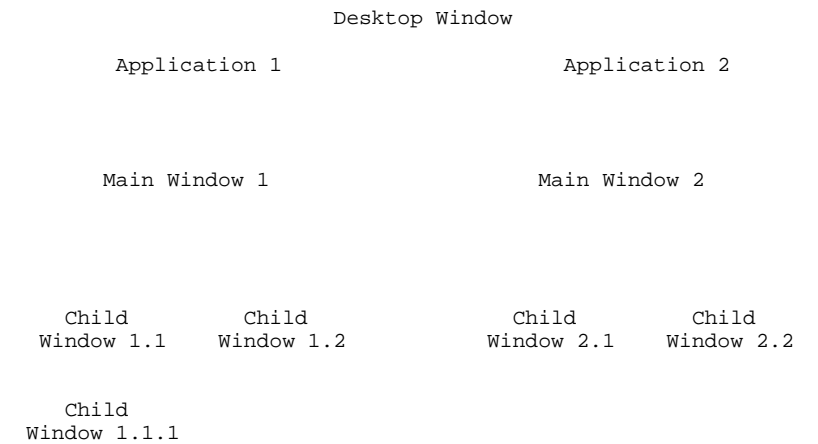
The *desktop-object window* is like a desktop window that is never displayed; it serves as the base window to coordinate the activity of an application's object windows. The desktop-object window cannot display windows nor process keyboard and mouse input. The primary purpose of the desktop-object window is to enable you to create windows that need not respond to messages at the same rate as the user interface.

Window Relationships

Window relationships define how windows interact with each other-on the screen and through messages. There are parent-child window relationships and window-owner relationships.

The *parent-child relationship* determines where and how windows appear when drawn on the screen. It also determines what happens to a window when a related window is destroyed or hidden. The parent-child rules apply to all windows at all times and cannot be modified.

Ownership determines how windows communicate using messages. Cooperating windows define and carry out their rules of ownership. Although some windows (such as windows of the preregistered public window class, WC_FRAME) have very complex rules of ownership, the application usually defines the ownership rules. The following figure represents the logical relationship of the windows in two applications.



Typical Window Relationships

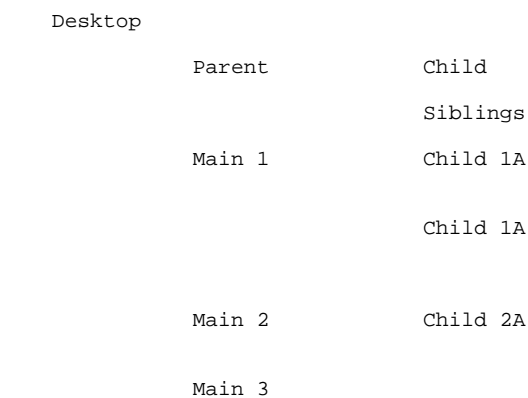
Parent-Child Relationship

Most windows have a *parent window*. (The exceptions are the desktop and desktop-object windows, which the system creates at system startup.) An application specifies the parent when it creates a window; then, the system uses the parent to determine where and how to draw any new windows, as well as when to *destroy* the windows (free all associated resources and remove the windows from the screen).

A *child window* is drawn relative to its parent. The coordinates given to specify the position of a window's lower-left corner are relative to the

lower-left corner of its parent. For example, a main window (child of the desktop) is drawn relative to the lower-left corner of the screen (the desktop window's lower-left corner).

All main windows are *siblings* because they share a common parent, the desktop window. Because sibling windows can overlap, an application or a user arranges the windows, one behind another (like a stack of papers on a desk), in the desired viewing order (called *z-order*). Z-order uses the desktop as a reference point for a "three-dimensional" ranking of the overlapping windows: the topmost window has the highest ranking, while the window at the bottom of the stack has the lowest ranking. The parent of the sibling windows is always at the bottom of the z-order. The following figure illustrates the hierarchy of such an arrangement.



Window Hierarchy

Although PM *supports* z-order, it does not *enforce* the expected appearance unless you specify the CS_CLIPCHILDREN or CS_CLIPSIBLINGS styles. No part of a child window ever appears outside the borders of its parent. If an application creates a window that is larger than its parent, or positions a window so that some or all of it extends beyond the borders of the parent, the extended portion of the child window is not drawn.

An application can use the WS_CLIPCHILDREN or WS_CLIPSIBLINGS styles to remove from a window's *clipping area* (the area in which the window can paint) the area occupied by its child or sibling windows. For example, an application can use these styles to prevent a window from painting over a child or sibling window containing a complex graphic that would be time-consuming to redraw.

When a window is minimized, hidden, or destroyed, all of its children are hidden, minimized, or destroyed as well. The order of destruction is always such that every window is destroyed before its parent. The window-destruction sequence starts at the bottom of descendency so that all related windows can be cleaned up; the last one to go is the window you asked to be destroyed. The final PM task in a window-destruction sequence is to send a WM_DESTROY message to that window, so it has one last chance to release any resources it has allocated and may still be holding.

Every window has only one parent, but can have any number of children. window in this tree is said to be a *descendant* of any window appearing above it in the branch, and an *ancestor* of any window appearing below it. There are two special cases, of course: the window immediately above is called the window's *parent*, and any window immediately below it is called its *child*. An application can change a window's parent window at any time by using the WinSetParent function. Changing the parent window also changes where and how the child window is drawn. The system displays the child within the borders of the new parent and draws the window according to the styles specified for the new parent.

Ownership

Any window can have an *owner window*. Typically, an application uses ownership to establish a connection between windows so that they can perform useful tasks together. For example, the title bar in an application's main window is owned by the frame window; but, together, the user can move the entire main window by clicking the mouse in the title bar and dragging. An application can set the owner window when it creates the window or at a later time.

Ownership establishes a relationship between windows that is independent of the parent-child relationship. While there are few predefined rules for owner- and owned-window interaction, a window *always* notifies its owner of anything considered a *significant event*.

The preregistered public window classes provided by the OS/2 operating system recognize ownership. Control windows of classes such as WC_TITLEBAR and WC_SCROLLBAR, notify their owners of events; frame windows, of class WC_FRAME, receive and process notification messages from the control windows they own. For example, a title-bar control sends a notification message to its owner when it receives a mouse click. If the owner is a frame window, it receives the notification message and prepares to move itself and its children.

Owner and owned windows must be created by the same thread; that is, they must belong to the same message queue. Because ownership is independent of the parent-child relationship, the owner and owned windows do not have to be descendants of the same parent window. However, this can affect how windows are destroyed. Destroying an owner window does not necessarily destroy an owned window. Except for frame windows, an application that needs to destroy an owned window that is not a descendant of the owner window must do so explicitly.

Frame windows sometimes own windows that are not descendants but, instead, are siblings. A frame window has the following special ownership properties:

- When the frame window is destroyed, it destroys all of the windows it owns, even if they are not descendants.
- When a frame window moves, the windows it owns move also. Owned windows that are not descendants maintain their positions, relative to the upper-left (not the usual lower-left) corner of the owner window. An owned window with the style `FS_NOMOVEWITHOWNER` does not move.
- When the frame window changes its position in the z-order, it changes the z-order of all the windows it owns.
- When the frame window is minimized or hidden, it hides all the windows it owns. Owned windows hidden this way are restored when the frame window is restored.

If an application needs this type of special processing for its own window classes, it must provide that support in the window procedures for those classes.

Note: Never create a window with an owner being a window in a different process. This causes problems when destroying the owned- or owner-window; usually the owned- or owner-window hangs.

Object Windows

Any descendant of the desktop-object window is called an *object window*. Typically, an application uses an object window to provide services for another window. For example, an application can use an object window to manage a shared database. In this way, a window can obtain information from the shared database by sending a message to and receiving a reply from the object window.

Only two system-defined messages are available to an object window-WM_CREATE and WM_DESTROY-but the object window enables the user to implement a set of user-defined messages. The window procedure for an object window does not have to process paint messages or user input. The object window processes only messages that affect the data belonging to the object.

HWND_OBJECT is the only identifier needed to create an object window. It is very unwise to create descendants of HWND_OBJECT in the same thread that creates descendants of HWND_DESKTOP: this causes the system to hang up or, at the very least, behave slowly. Object windows, sometimes referred to as *orphan* windows, require no owner.

The rules for parent-child and ownership relationships also apply to object windows. In particular, changing the parent window of an object window to the desktop window, or to a descendant of the desktop window, causes the system to display the object window if the object window has the WS_VISIBLE style.

Application Windows

An application can use several types of *secondary* windows: frame windows, client windows, control windows, dialog windows, message boxes, and menus. Typically, an application's main window consists of several of these windows acting as one.

A *frame window* is a window that an application uses as the base when constructing a main window or other composite window, such as a dialog window or message box. (A *composite window* is a collection of windows that interact with one another and are kept together as a unit.) A frame window provides basic features, such as borders and a menu bar. Frame windows have a set of resources associated with them. These include icons, menus, and accelerators (*shortcut keys* to the user), which, typically, are defined in an application's resource file.

A *dialog window* is a frame window that contains one or more control windows. Dialog windows are used almost exclusively for prompting the user for input. An application usually creates a dialog window when it needs additional information to complete a command. The application destroys the dialog window after the user has provided the requested information.

A *message box* is a frame window that an application uses to display a note, caution, or warning to the user. For instance, an application can use a message box to inform the user of a problem that the application encountered while performing a task.

A *client window* is the window in which the application displays the current document or data. For example, a desktop-publishing application displays the current page of a document in a client window. Most applications create at least one client window. The application must provide a function, called a *window procedure*, to process input to the client window and to display output.

A *control window* is a window used in conjunction with another window to perform useful tasks, such as displaying a menu or scrolling information in a client window. The operating system provides several predefined control-window classes that an application can use to create control windows. Control windows include buttons, entry fields, list boxes, combination boxes, menus, scroll bars, static text, and title bars.

A *menu* is a control window that presents a list of commands and other menus to the user. Using a mouse or the keyboard, the user can select a task; the application then performs the selected task.

Window Input and Output

The user directs input data to windows from a mouse and the keyboard. Keyboard input goes to the window with *input focus*, and, normally, mouse input goes to the window under the mouse pointer.

Windows also are places to display output data. PM uses windows to display text and graphics on the screen and to process input from the mouse and keyboard. Windows provide the same input and output capabilities as a virtual graphics terminal without having direct control of the hardware.

An application is responsible for painting the data for the window classes it registers and creates. This data can be graphics text or pictures or fixed-size alphanumeric text. Normally it is not necessary for the application to paint the system-provided window classes; the OS/2 window procedures for those window classes do the painting.

Active Window and Focus Window

All frame-window ancestors of the input focus window are said to be *active*, meaning that the user interacts with them. The active window usually is the topmost main window, which is positioned above all other top-level windows on the screen. The active window is indicated by some form of highlighting. For example, a highlighted title bar shows that a standard frame window is active; an active dialog window has a highlighted border. These types of highlighting ensure that the user can see the window that is accepting input.

A main window (or one of its child windows) is activated by using a mouse or the keyboard. When a window is activated, it receives a WM_ACTIVATE message with its first parameter set to TRUE. When it is deactivated, it receives a WM_ACTIVATE message with its first parameter set to FALSE.

The *focus window* can be the active window or one of its descendant windows. The user can change the *input focus* the same way active windows are changed-by mouse or keyboard. However, the application has more control over the input focus. For example, in a window containing several text entry fields, the tab keys can move the input focus from one input field to another. A WM_SETFOCUS message is sent to the window procedure when a window is gaining or losing the input focus. The WinQueryFocus function tells the user which window has the input focus.

Messages

Messages are a fundamental part of the operating system. PM applications use messages to communicate with the operating system and one another. The system uses messages to communicate with applications to ensure concurrent running and sharing of devices. Typically, a message notifies the receiving application that an *event* has occurred. The operating system identifies the appropriate application window to receive a message by the window handle included in the message. Sources of events that cause messages to be issued to applications are the user, the operating system, the application, or another application.

The User

Mouse or keyboard input to an application window causes the operating system to direct messages to that window.

The Operating System

Managing the application windows on the screen, the operating system issues messages to the windows, usually as an indirect result of user interaction. These messages enable the system to work in a uniform and well-ordered manner. For example, where several application windows overlap, and the user terminates an application so that its window disappears, the operating system issues messages to the underlying application windows so that they can repaint themselves.

The Application

An event can occur in the application to which another part of that application should respond; for example, when the contents of its window no longer accurately reflect the status of the application. The application can define its own messages outside the range of system-defined messages to communicate such events.

Another Application

Communication with other applications through the operating system ensures cooperative use of the system; it even can be used to exchange data. For example, an arithmetic application can supply the results of a lengthy calculation to a business graphics application.

Enabled and Disabled Windows

An application uses the `WinEnableWindow` function to enable or disable window input. By default, a window is enabled when it is created. However, an application can disable a newly created window.

An application usually disables a window to prevent the user from using the window. For example, an application might disable a push button in a dialog window. Enabling a window restores normal input; an application can enable a disabled window at any time.

When an application uses the `WinEnableWindow` function to disable an existing window, that window also loses keyboard focus. `WinEnableWindow` sets the keyboard focus to `NULL`, which means that no window has the focus. If a child window or other descendant window has the keyboard focus, it loses the focus when the parent window is disabled.

An application can determine whether a window is enabled by calling `WinIsWindowEnabled`.

System-Modal Window

An application can designate a *system-modal window*: a window that receives all keyboard and mouse input, effectively disabling all other windows. The user must respond to the system-modal window before continuing work in other windows. An application sets and clears the

system-modal window by using the WinSetSysModalWindow function.

Because system-modal windows have absolute control of input, you must be careful when using them in your applications. Ideally, an application uses a system-modal window only when there is danger of losing data if the user does not respond to a problem immediately.

Although an application can destroy a system-modal window, the new active window then becomes a system-modal window. An application can make another window active while the first system-modal window exists. But again, the new active window will become the system-modal window. In general, once a system-modal window is set, it continues to exist in the PM session until the application explicitly clears it.

Window Creation

Before any thread in an application can create windows, it must:

1. Call WinInitialize to create an anchor block
2. Call WinCreateMsgQueue to create a message queue for the thread

Then, it can create one or more windows by calling one of the window-creation functions, such as WinCreateWindow.

The window-creation functions require that the following information be supplied in some form:

- Class
- Styles
- Name
- Parent window
- Position relative to the parent window
- Position relative to any sibling windows (z-order)
- Dimensions
- Owner window
- Identifier
- Class-specific data
- Resources

Every window belongs to a *window class* that defines that window's appearance and behavior. The chief component of the window class is the *window procedure*. The window procedure is the function that receives and processes all messages sent to the window.

Every window has a *style*. The window style specifies aspects of a window's appearance and behavior that are not specified by the window's class. For example, the WC_FRAME class always creates a frame window, but the FS_BORDER, FS_DLGBOARDER, and FS_SIZEBOARDER styles determine the style of a frame window's border. A few window styles apply to all windows, but most apply only to windows of specific window classes. The window procedure for a given class interprets the style and allows an application to adapt a window of a given class for a special circumstance. For example, an application can give a window the style WS_SYNCPOINT to cause it to be painted immediately whenever any portion of the window becomes invalid. Normally, a window is painted only if there are no messages waiting in the message queue.

A window can have a text string associated with it. Typically, the window text is displayed in the window or in a title bar. The class of window determines whether the window displays the text and, if so, where the text appears within the window.

Every window except the desktop window and desktop-object window has a *parent window*. The parent provides the coordinate system used to position the window and also affects aspects of a window's appearance. For example, when the parent window is minimized, hidden, or destroyed, the parent's child windows are minimized, hidden, or destroyed also.

Every window has a screen position, size, and z-order position. The *screen position* is the location of the window's lower-left corner, relative to the lower-left corner of its parent window. A window's size is its width and height, measured in pels. A window's *z-order position* is the position of the window in the order of overlapping windows. This viewing order is oriented along an imaginary axis, the *z* axis, extending outward from the screen. The window at the top of the z-order overlaps all *sibling* windows (that is, windows having the same parent window). A window at the bottom of the z-order is overlapped by all sibling windows. An application sets a window's z-order position by placing it behind a given sibling window or at the top or bottom of the z-order of the windows.

A window can own, or be owned by, another window. The owner-owned relationship affects how messages are sent between windows, allowing an application to create combinations of windows that work together. A window issues messages about its state to its owner window; the owner window issues messages back about what action to perform next.

The *window handle* is a unique number across the system that is totally unambiguous—it identifies one particular window in the system and is assigned by the system. A *window identifier* is analogous to a "given" name in family relationships; the only requirement is that the name be unique among siblings.

A window can have class-specific data that further defines how the window appears and behaves when it is created. The system passes the

class-specific data to the window procedure, which then applies the data to the new window.

Window-Creation Functions

The basic window-creation function is `WinCreateWindow`. This function uses information about a window's class, style, size, and position to create a new window. All other window-creation functions, such as `WinCreateStdWindow` and `WinCreateDlg`, supply some of this information by default and create windows of a specific class or style.

Although the `WinCreateWindow` function provides the most direct means of creating a window, most applications do not use it. Instead, they often use the `WinCreateStdWindow` function to create a main window and the `WinDlgBox` or `WinCreateDlg` functions to create dialog windows.

The `WinCreateMenu`, `WinLoadMenu`, `WinLoadDlg`, `WinMessageBox`, and `WinCreateFrameControls` functions also create windows. Each of these functions substitutes for one or more required calls to `WinCreateWindow` to create a given window. For example, an application can create a frame window, one or more control windows, and a client window in a single call to `WinCreateStdWindow`.

Window-Creation Messages

While creating a window, the system sends messages to that window's window procedure. The window procedure receives a `WM_CREATE` message, saying that the window is being created. The window also receives a `WM_ADJUSTWINDOWPOS` message, specifying the initial size and position of the window being created. This message lets the window procedure adjust the size and position of the window before the window is displayed.

The system also sends other messages while creating a window; the number and order of these messages depend on the class and style of the window and the function used to create it.

Window Classes

Each window of a specific window class uses the window procedure associated with that class. An application can create one or more windows that belong to the same window class. Because each window of the same class is processed by the same window procedure, they all behave the same way. Since many windows can result from one window procedure, coding overhead is greatly reduced. There are two types of window classes: public and private.

Public Window Classes

A *public window class* is one that has a reentrant window procedure that is registered and resides in a dynamic link library (DLL); it can be used by any process in the system to create windows. The operating system provides several preregistered public window classes. You can specify the system-provided window classes by using the symbolic identifiers that have the prefix `WC_`, as shown in the following table:

Class Name	Description
<code>WC_BUTTON</code>	Consists of buttons and boxes the user can select by clicking the pointing device or using the keyboard.
<code>WC_CONTAINER</code>	Creates a control for the user to group objects in a logical manner. A container can display those objects in various formats or views. The container control supports drag and drop so the user can place information in a container by simply dragging and dropping.

<code>WC_ENTRYFIELD</code>	Consists of a single line of text that the user can edit.
<code>WC_FRAME</code>	A window class that can contain child windows of many of the other window classes.
<code>WC_LISTBOX</code>	Presents a list of text items from which the user can make selections.
<code>WC_MENU</code>	Presents a list of items that can be displayed horizontally as menu bars, or vertically as pull-down menus. Menus usually are used to provide a command interface to applications.
<code>WC_NOTEBOOK</code>	Creates a control for the user that is displayed as a number of pages. The top page is visible, and the others are hidden, with their presence being indicated by a visible edge on each of the back pages.
<code>WC_SCROLLBAR</code>	Lets the user scroll the contents of an associated window.
<code>WC_SLIDER</code>	Creates a control that is usable for producing approximate (analog) values or properties. Scroll bars were used for this function in the past, but the slider provides a more flexible method of achieving the same result, with less programming effort.
<code>WC_SPINBUTTON</code>	Creates a control that presents itself to the user as a scrollable ring of choices, giving the user quick access to the data. The user is presented only one item at a time, so the spin button should be used with data that is intuitively related.
<code>WC_STATIC</code>	Simple display items that do not respond to keyboard or pointing device events.
<code>WC_TITLEBAR</code>	Displays the window title or caption and lets the user move the window's owner.
<code>WC_VALUESET</code>	Creates a control similar in function to the radio buttons but provides additional flexibility to display graphical, textual, and numeric formats. The values set with this control are mutually exclusive.

With the exception of `WC_FRAME`, the system-provided window classes are known as *control window classes* because they give the user an easy means of controlling specific types of interaction. For example, the `WC_BUTTON` class allows single or multiple selections. These windows conform to the IBM* Systems Application Architecture (SAA) Common User Access (CUA) definition. They are designed specifically to provide function that meets the needs for a graphics-based standard user interface. The code fragments provided in this guide make extensive use of the system window classes.

Private Window Classes

A *private window class* is one that an application registers for its own use; it is available only to the process that registers it. The application-provided window procedure for a private window class resides either in the application's executable files or in a DLL file. A private window class is deleted when its registering process is terminated.

Window Styles

A window can have a combination of styles; an application can combine styles by using the bitwise inclusive OR operator. An application

usually sets the window styles when it creates the window. The OS/2 operating system provides several standard window styles that apply to all windows. It also provides many styles for the predefined frame and control windows. The frame and control styles are unique to each predefined window class and can be used only for windows of the corresponding class.

Initially, the styles of the window class used to create the window determine the styles of the new window. For example, if the window class has the style CS_SYNCPAINT, all windows created using that class, by default, will have the window style WS_SYNCPAINT. The OS/2 operating system has the following standard window styles:

Style Name	Description
WS_CLIPCHILDREN	Prevents a window from painting over its child windows. This style increases the time necessary to calculate the visible region. This style is usually not necessary because if the parent and child windows overlap and both are invalidated, the system draws the parent window before drawing the child window. If the child window is invalidated independently of the parent window, the system redraws only the child window. If the update region of the parent window does not intersect the child window, drawing the parent window causes the child window to be redrawn. This style is useful to prevent a child window that contains a complex graphic from being redrawn unnecessarily. WS_CLIPCHILDREN is an absolute requirement if a window with children ever performs output in response to any message other than WM_PAINT. Only WM_PAINT processing is synchronized such that the children will get their messages after the parent.
WS_CLIPSIBLINGS	Prevents a window from painting over its sibling windows. This style protects sibling windows but increases the time necessary to calculate the visible region. This style is appropriate for windows that overlap and that have the same parent window.
WS_DISABLED	Used by an application to disable a window. It is up to the window to recognize this style and reject input.
WS_GROUP	Specifies the first control of a group of controls in which the user can move from one control to the next by using the ARROW keys. All controls defined after the control with the WS_GROUP style belong to the same group. The next control with the WS_GROUP style ends the first group and starts a new group.
WS_MAXIMIZED	Enlarges a window to the maximum size.
WS_MINIMIZED	Reduces a window to the size of an icon.
WS_PARENTCLIP	Extends a window's visible region to include that of its parent window. This style simplifies the calculation of the child window's visible region but is potentially dangerous because the parent window's visible region is usually larger than the child window.
WS_SAVEBITS	Saves the screen area under a window as a bit map. When the user hides or moves the window, the system restores the image by copying the bits; there is no need to add the area to the uncovered window's update region. The style can improve system performance but also can consume a great deal of memory. It is recommended only for transient windows, such as menus and dialog windows, not for main application windows.
WS_SYNCPAINT	Causes a window to receive WM_PAINT messages immediately after a part of the window

becomes invalid. Without this style, the window receives WM_PAINT messages only if no other message is waiting to be processed.

WS_TABSTOP

Specifies one of any number of controls through which the user can move by tabbing. Pressing the TAB key moves the keyboard focus to the next control that has the WS_TABSTOP style.

WS_VISIBLE

Makes a window visible. The operating system draws the window on the screen unless overlapping windows completely obscure it. Windows without this style are hidden. If overlapping windows completely obscure the window, the window is still considered visible. (*Visibility* means that the operating system draws the window if it can.)

Window Handles

After creating a window, the creation function returns a window handle that uniquely identifies the window. An application can use this handle to direct the action of functions to the window. Window handles have the data type HWND; applications must use this data type when declaring variables that hold window handles.

There are special constants that an application can use instead of a window handle in certain functions. For example, an application can use HWND_DESKTOP in the WinCreateWindow function to specify the desktop window as the new window's parent. Similarly, HWND_OBJECT represents the desktop-object window. HWND_TOP and HWND_BOTTOM represent the top and bottom positions relative to the z-order position of a window.

Although the NULL constant is not a window handle, an application can use it in some functions to specify that no window is affected. For example, an application can use NULL in the WinCreateWindow function to create a window that has no owner window. Some functions might return NULL, indicating that the given action applies to no window.

Window Size and Position

A window's size and position can be expressed as a bounding rectangle, given in coordinates relative to its parent. An application specifies the window's initial size and position when creating the window.

To use the system-default values for the initial size and position of a frame window, an application can specify the FCF_SHELLPOSITION frame-creation flag. The application can change a window's size and position at any time.

Note:

The default coordinate system for a window specifies that the point (0,0) is at the lower-left corner of the window, with coordinates increasing as they go upward and to the right.

A window can be positioned anywhere in relation to its parent.

Size

A window's *size* (width and height) is given in pels, in the range 0 through 65535. A window can have 0 width and height; however, a window with 0 width or height is not drawn on the screen, even though it has the WS_VISIBLE style.

An application can create very large windows; however, it should check the size of the screen before enlarging a window size. One way to choose an appropriate size is to use the `WinGetMaxPosition` function to retrieve the size of the maximized window. A window that is larger than its maximized size will be larger than the screen also.

An application can retrieve the current size of the window by using the `WinQueryWindowRect` function.

Position

A window's *position* is defined as the x,y coordinates of its lower-left corner. These coordinates, sometimes called *window coordinates*, always are relative to the lower-left corner of the parent window. For example, a window having the coordinates (10,10) is placed 10 pels to the right of, and 10 pels up from, the lower-left corner of its parent window. Notice, however, that a window can be positioned anywhere in relation to its parent, but always relative to the parent's lower-left corner.

Adjusting a window's position can improve drawing performance. For example, an application could position a window so that its horizontal position is a multiple of 8, relative to the screen *origin* (the lower-left corner of the screen). Coordinates that are multiples of 8 correspond to byte boundaries in the screen-memory bit map. It is usually faster to start drawing at a byte boundary.

By default, the system positions a frame window on a byte boundary; but an application can override this action by using the `FCF_NOBYTEALIGN` style when creating the window.

Size and Position Messages

A window receives messages when it changes size or position. Before a change is made, the system might send a `WM_ADJUSTWINDOWPOS` message to allow the window procedure to make final adjustments to the window's size and position. This message includes a pointer to an `SWP` structure that contains the requested width, height, and position. If the window procedure adjusts these values in the structure, the system uses the adjusted values to redraw the window. The `WM_ADJUSTWINDOWPOS` message is not sent if the change is a result of a call to the `WinSetWindowPos` function with the `SWP_NOADJUST` constant specified.

After a change has been made to a window, the system sends a `WM_SIZE` message to specify the new size of the window. If the window has the class style `CS_MOVENOTIFY`, the system also sends a `WM_MOVE` message, which includes the new position for the window. The system sends a `WM_SHOW` message if the visibility of the window has changed.

System Commands

An application that has a window with a system menu can change the size and position of that window by sending system commands. The system commands are generated when the user chooses commands from the system menu. An application can emulate the user action by sending a `WM_SYSCOMMAND` message to the window.

Following are some of the system commands:

Command	Description
<code>SC_SIZE</code>	Starts a Size command. The user can change the size of the window with a mouse and the keyboard.
<code>SC_MOVE</code>	Starts a Move command. The user can move the window with a mouse and the keyboard.
<code>SC_MINIMIZE</code>	Minimizes the window.
<code>SC_MAXIMIZE</code>	Maximizes the window.
<code>SC_RESTORE</code>	Restores a minimized or maximized window to its previous size and position.

`SC_CLOSE` Closes the window. This command sends a `WM_CLOSE` message to the window. The window performs all tasks needed to clean up and destroy itself.

Window Data

Every window has an associated data structure. The window data structure contains all the information specified for the window at the time it was created and any additional information supplied for the window since that time. Although the exact size and meaning of the information in the window data structure are private to the system, an application can access any of the following data items via system-provided functions:

- Pointer to window-instance data structure
- Pointer to window procedure
- Parent-window handle
- Owner-window handle
- Handle of first child window
- Handle of next sibling window
- Window size and position (expressed as a rectangle)
- Window style
- Window identifier
- Update-region handle
- Message-queue handle

An application can examine and modify this data by using functions such as `WinQueryWindowUShort` and `WinSetWindowUShort`. These functions let an application access data that is stored as 16-bit integers. Other functions let an application access data containing 32-bit integers and pointers. Several functions indirectly affect the data items in the window data structure. For example, the `WinSubclassWindow` function replaces the window-procedure pointer, and the `WinSetWindowPos` function changes the size and position of the window.

An application can extend the number of available data items in the window data structure by specifying a count of extra bytes when it registers the corresponding window class. Then, the window procedure can use these bytes to store information about the window. The `WinQueryWindowUShort` and `WinSetWindowUShort` functions give direct access to the extra bytes.

It generally is not a good idea to use direct storage in the window data. It is better to allocate a data structure dynamically and set a pointer to that data structure in the window words. This provides two advantages:

1. Most importantly, it is a symbolic way of referencing the data structure. It is very easy to make mistakes and provide the wrong offsets to `WinQueryWindowUShort` and so forth.
 2. You now can add and remove fields without cross dependencies because you now use *symbolic* references; whereas, when you use the technique of putting window words directly in the window data structure, you have to account for changed offsets.
-

Window Resources

Window resources are read-only data segments stored in an application's EXE file or in a dynamic link library's DLL file. Predefined PM window resources include keyboard accelerator tables, icons, menus, bit maps, dialog boxes, and so forth; these are not a regular part of the application window's code and data. Because, in most cases, window resources are not loaded into memory when the operating system runs a program, the resources can be shared by multiple instances of the same application.

Most window resources are stored in a format that is unique to each resource type. The application does not need to know these formats because the system translates them, as necessary, for use in PM functions. The following table lists the ten most commonly used PM window resource types.

Resource Identifier	Description
<code>RT_ACCELTABLE</code>	Keyboard accelerator table

RT_BITMAP	Bit map
RT_DIALOG	Dialog box template
RT_FONT	Font
RT_FONTPATH	Font directory
RT_MENU	Menu template
RT_MESSAGE	Message string
RT_POINTER	Icon or mouse
RT_RCDATA	Programmer-defined data
RT_STRING	Text string

To access these resources, you must prepare a *resource file* (ASCII file with the extension .RC). Then the ASCII resource file must be compiled into binary images using the resource compiler. The compiled resource file extension is RES; it can be linked into your program's EXE file or to a dynamic link library's DLL file.

Maximized and Minimized Windows

A *maximized window* is a window that has been enlarged to fill the screen. Although a window's size can be set so that it fills the screen exactly, a maximized window is slightly different: the system automatically moves the window's title bar to the top of the screen and sets the WS_MAXIMIZED style for the window.

A *minimized window* is a window whose size has been reduced to exactly the size of an icon *or*, in the Workplace Shell*, it disappears altogether (by default). Like a maximized window, a minimized window is more than just a window of a given size; typically, the system moves the (icon) minimized window to the lower part of the screen and sets the WS_MINIMIZED style for that window. The lower part of the screen is sometimes called the *icon area*. Unless the application specifies another position, the system moves a minimized window into the first available icon position in the icon area.

If a window is created with the WS_MAXIMIZED or WS_MINIMIZED styles, the system draws the window as a maximized or minimized window.

An application can restore maximized or minimized windows to their previous size and position by specifying the SWP_RESTORE flag in a call to the WinSetWindowPos function.

Window Visibility

A window that is a descendant of the desktop window can be either visible or invisible. The system displays a visible window on the screen. It hides an invisible window by not drawing it. If a window is visible, the user can supply input to the window and view the window's output. If a window is invisible, the window, in effect, is disabled. An invisible window can process messages from the system or from other windows, but it cannot process user input or display output. An application sets a window's visibility state when it creates the window. Later, a user or the application can change the visibility state.

The visible region of a window is the position clipped by any overlapping windows. These overlapping windows can be child windows or other main windows in the system. The visible region is defined by a set of one or more rectangles, as shown in the following figure.

A

B

C

D

- Visible region for Window A

Visible Region for Window A

A window is visible if the `WS_VISIBLE` style is set for the window. By default, the `WinCreateWindow` function creates invisible windows unless the application specifies `WS_VISIBLE`. The application often hides a window to keep its operational details from the user. For example, an application can keep a new window invisible while it customizes the window's appearance. An application can determine whether a window has the `WS_VISIBLE` style by using the `WinIsWindowVisible` function.

Even if a window has the `WS_VISIBLE` style, the user might not be able to see the window on the screen because other windows completely overlap it, or it might have been moved beyond the edge of its parent. A visible window is subject to the clipping rules established by its parent-child relationship. If the window's parent window is not visible, the window will not be visible. Because a child window is drawn relative to its parent's lower-left corner, if the parent window is moved beyond the edge of the screen, the child window also will be moved. In other words, if a user moves the parent window containing the child window far enough off the edge of the screen, the user will not be able to see the child window, even though the child window and its parent window have the `WS_VISIBLE` style. To determine whether the user actually can see a window, an application can use the `WinIsWindowShowing` function.

Window Destruction

In general, an application must destroy all the windows it creates. It does this by using the `WinDestroyWindow` function. When a window is destroyed, the system hides the window, if it is visible, and then removes any internal data associated with the window. This invalidates the window handle so that it can no longer be used by the application.

An application destroys many of the windows it creates soon after creating them. For example, an application usually destroys a dialog window as soon as the application has sufficient input from the user to continue its task. An application eventually destroys the main window of the application (before terminating).

Destroying a window does not affect the window class from which the window was created. New windows still can be created using that class, and any existing windows of that class continue to operate.

When the application calls `WinDestroyWindow`, the system searches the descendency tree for all windows below the specified window and destroys them from the bottom up, so each child receives `WM_DESTROY` before its parent. Each destroyed window is responsible for cleaning up its own resources in response to the `WM_DESTROY` message.

If a presentation space was created by the `WinGetPS` function for any of the windows to be destroyed, it must be released by calling the `WinReleasePS` function. The application must do this before calling the `WinDestroyWindow` function. If a presentation space is associated with the device context for the window, the application must disassociate or destroy the presentation space by using the `GpiAssociate` or `GpiDestroyPS` function before calling `WinDestroyWindow`. Failing to release a resource can cause an error.

For more information about presentation spaces and device contexts, see [Painting and Drawing](#).

If the window being destroyed is the active window, both the active and focus states are transferred to another window. The window that becomes the active window is the next window, as determined by the `Alt+Esc` key combination. The new active window then determines which window receives the keyboard focus.

Using Windows

The following sections explain how to create and use windows in an application, how to manage ownership and parent-child window relationships, and how to move and size windows.

Creating a Top-Level Frame Window

The main window in most applications is a top-level frame window. An application creates a top-level frame window by specifying the handle of the desktop window, or `HWND_DESKTOP`, as the `hwndParent` parameter in a call to the `WinCreateStdWindow` function.

The following figure shows the `main()` function for a simple PM application. This function initializes the application, creates a message queue, and registers the window class for the client window before creating a top-level frame window.

```
#define IDR_RESOURCES 1

MRESULT EXPENTRY ClientWndProc(HWND, ULONG, MPARAM, MPARAM);

int main(VOID)
{
    HWND hwndFrame;
    HWND hwndClient;
    HMQ hmq;
    QMSG qmsg;
    HAB hab;

    /* Set the frame-window creation flags. */
    ULONG flFrameFlags =
        FCF_TITLEBAR      /* Title bar */
        FCF_SIZEBORDER    /* Size border */
        FCF_MINMAX        /* Minimize and maximize buttons. */
        FCF_SYSMENU       /* System menu */
        FCF_SHELLPOSITION /* System-default size and position */
        FCF_TASKLIST;     /* Add name to Task List. */

    /* Initialize the application for PM */
    hab = WinInitialize(0);

    /* Create the application message queue. */
    hmq = WinCreateMsgQueue(hab, 0);

    /* Register the class for the client window. */
    WinRegisterClass(
        hab, /* Anchor block handle */
        "MyPrivateClass", /* Name of class being registered */
        (PFNWP)ClientWndProc, /* Window procedure for class */
        CS_SIZEREDRAW | /* Class style */
        CS_HITTEST, /* Class style */
        0); /* Extra bytes to reserve */

    /* Create a top-level frame window with a client window
    /* that belongs to the window class "MyPrivateClass". */
    hwndFrame = WinCreateStdWindow(
        HWND_DESKTOP, /* Parent is desktop window. */
        WS_VISIBLE, /* Make frame window visible. */
        &flFrameFlags, /* Frame controls */
        "MyPrivateClass", /* Window class for client */
        NULL, /* No window title */
        WS_VISIBLE, /* Make client window visible. */
        (HMODULE) 0, /* Resources in application module */
        IDR_RESOURCES, /* Resource identifier */
        NULL); /* Pointer to client window handle */

    /* Start the main message loop. Get messages from the
    /* queue and dispatch them to the appropriate windows. */
    while (WinGetMsg(hab, &qmsg, 0, 0, 0))
        WinDispatchMsg(hab, &qmsg);

    /* Main loop has terminated. Destroy all windows and the
    /* message queue; then terminate the application. */
    WinDestroyWindow(hwndFrame);
    WinDestroyMsgQueue(hmq);
    WinTerminate(hab);

    return 0;
}
```

Creating an Object Window

An application can create an object window by using the `WinCreateWindow` function and setting the desktop-object window as the parent window. The code fragment in the following figure shows how to create an object window.

```
#define ID_OBJWINDOW 2

HWND hwndObject;

hwndObject = WinCreateWindow(
    HWND_OBJECT,      /* Parent is object window. */
    "MyObjClass",     /* Window class for client */
    NULL,             /* Window text */
    0,                /* No styles for object window */
    0, 0,             /* Lower-left corner */
    0, 0,             /* Width and height */
    NULL,             /* No owner */
    HWND_BOTTOM,      /* Inserts window at bottom of z-order */
    ID_OBJWINDOW,     /* Window identifier */
    NULL,             /* No class-specific data */
    NULL);            /* No presentation data */
```

Querying Window Data

An application can examine the values in the data structure associated with a window by using the `WinQueryWindowUShort` and `WinQueryWindowULong` functions. Each of these functions specifies a structure data item to examine. The index value can be an integer representing a zero-based byte index or a constant (`QWS_`) that identifies a specific item of data. The code fragment in the following figure obtains the programmer-defined identifier of the object window defined in the previous example:

```
HWND hwndObject;
USHORT usObjID;

usObjID = WinQueryWindowUShort(hwndObject, QWS_ID);
```

Changing the Parent Window

An application can change a window's parent window by using the `WinSetParent` function. For example, in an application that uses child windows to display documents, you might want only the active document window to show a system menu. You can do this by changing that menu's parent window back and forth between the document window and the object window when `WM_ACTIVATE` messages are received. This technique is shown in the code fragment in the following figure.

```
switch (msg) {

case WM_ACTIVATE: {

HWND hwndFrame, hwndSysMenu, hwnd;

    /* Get the handles of the frame window and system menu. */
    hwndFrame = WinQueryWindow(hwnd, QW_PARENT);
    hwndSysMenu = WinWindowFromID(hwndFrame, FID_SYSMENU);

    /* If the window is being activated, make the frame window the */
```

```

/* parent of the system menu. Otherwise, hide the system menu */
/* by making the object window the parent. */

if ( SHORT1FROMMP(mpl))
    WinSetParent(hwndSysMenu, hwndFrame, TRUE);

else
    WinSetParent(hwndSysMenu, HWND_OBJECT, TRUE);

}

return 0;
}

```

Finding a Parent, Child, or Owner Window

An application can determine the parent, child, and owner windows for any window by using the WinQueryWindow function. This function returns the window handle of the requested window.

The code fragment in the following figure determines the parent window of the given window:

```

HWND hwndParent;
HWND hwndMyWindow;

hwndParent = WinQueryWindow(hwndMyWindow, QW_PARENT);

```

The code fragment in the following figure determines the *topmost* child window (the child window in the top z-order position):

```

HWND hwndTopChild;
HWND hwndParent;

hwndTopChild = WinQueryWindow(hwndParent, QW_TOP);

```

If a given window does not have an owner or child window, WinQueryWindow returns NULL.

Setting an Owner Window

An application can set the owner for a window by using the WinSetOwner function. Typically, after setting the owner, a window notifies the owner window of the new relationship by sending it a message.

The code fragment in the following figure shows how to set the owner window and send it a message:

```

#define NEW_OWNER 1

HWND hwndMyWindow;
HWND hwndNewOwner;

if (WinSetOwner(hwndMyWindow, hwndNewOwner))

    /* Send a notification message. */
    WinSendMsg(hwndNewOwner, /* Sends to owner */
               WM_CONTROL, /* Control message for notification */
               (MPARAM) NEW_OWNER, /* Notification code */
               0);

```

```
NULL); /* No extra data */
```

A window can have only one owner, so WinSetOwner removes any previous owner.

Retrieving the Handle of a Child or Owned Window

A parent or owner window can retrieve the handle of a child or owned window by using the WinWindowFromID function and supplying the identifier of the child or owned window. WinWindowFromID searches all child and owned windows to locate the window with the given identifier. The window identifier is set when the application creates the child or owned window.

Typically, an owned window uses WinQueryWindow to get the handle of the owner window; then uses WinSendMessage to issue a notification message to its owner window.

The code fragment in the following figure retrieves the window handle of an owner window and sends the window a WM_ENABLE message.

```
HWND hwndOwned;
HWND hwndOwner;

case WM_CONTROL:
    switch (SHORT2FROMMP (mp2)) {
        case BN_CLICKED:
            hwndOwned = WinWindowFromID(hwndOwner,
                (ULONG)SHORT1FROMMP(mp1));
            WinSendMessage(hwndOwned, WM_ENABLE,
                (LPARAM)TRUE, (LPARAM) NULL);
            return 0;
        . /* Check for other notification codes. */
        .
    }
}
```

An application also can retrieve the handle of a child window by using the WinWindowFromPoint function and supplying a point in the corresponding parent window.

Enumerating Top-Level Windows

An application can enumerate all top-level windows in the system by using the WinBeginEnumWindows and WinGetNextWindow functions. An application also can create a list of all child windows for a given parent window using WinBeginEnumWindows. This list contains the window handles of immediate child windows. By using WinGetNextWindow, the application then can retrieve the window handles, one at a time, from the list. When the application has finished using the list, it must release the list with the WinEndEnumWindows function.

The code fragment in the following figure shows how to enumerate all top-level windows (all immediate child windows of the desktop window):

```
HWND hwndTop;
HENUM henum;

/* Enumerate all top-level windows. */

henum = WinBeginEnumWindows(HWND_DESKTOP);

/* Loop through all enumerated windows. */
while (hwndTop = WinGetNextWindow(henum)) {
    . /* Perform desired task on each window. */
    .
}
}
```

```
WinEndEnumWindows(henum);
```

Moving and Sizing a Window

An application can move a window by using the `WinSetWindowPos` function and specifying the `SWP_MOVE` constant. The function changes the position of the window to the specified position. The position is always given in coordinates relative to the parent window.

The code fragment in the following figure moves the window to the position (10,10):

```
HWND hwnd;  
  
WinSetWindowPos(  
    hwnd,                /* Window handle */  
    NULL,                /* Not used for moving and sizing */  
    10, 10,              /* New position */  
    0, 0,                /* Not used for moving */  
    SWP_MOVE);           /* Move window */
```

An application can set the size of a window by using the `WinSetWindowPos` function and specifying the `SWP_SIZE` constant. `WinSetWindowPos` changes the width and height of the window to the specified width and height.

An application can combine moving and sizing in a single function call, as shown in the following figure.

```
HWND hwnd;  
  
WinSetWindowPos(  
    hwnd,                /* Window handle */  
    NULL,                /* Not used for moving and sizing */  
    10, 10,              /* New position */  
    200, 200,            /* Width and height */  
    SWP_MOVE | SWP_SIZE); /* Move and size window. */
```

An application can retrieve the current size and position of a window by using the `WinQueryWindowPos` function. This function copies the current information to an `SWP` structure.

The code fragment in the following figure uses the current size and position to change the height of the window, leaving the width and position unchanged.

```
HWND hwnd;  
SWP swp;  
  
WinQueryWindowPos(hwnd, &swp);  
WinSetWindowPos(  
    hwnd,                /* Window handle */  
    NULL,                /* Not used for moving and sizing */  
    0, 0,                /* Not used for sizing */  
    swp.cx,              /* Current width */  
    swp.cy + 200,        /* New height */  
    SWP_SIZE);           /* Change the size. */
```

An application also can move and change the size of several windows at once by using the `WinSetMultWindowPos` function. This function takes an array of `SWP` structures. Each structure specifies the window to be moved or changed.

An application can move and size a window even if it is not visible, although the user is not able to see the effects of the moving and sizing until the window is visible.

Redrawing Windows

When the system moves a window or changes its size, it can invalidate all or part of that window. The system attempts to preserve the contents of the window and copy them to the new position; however, if the window's size is increased, the window must fill the area exposed by the size change. If a window is moved from behind an overlapping window, any area formerly obscured by the other window must be drawn. In these cases, the system invalidates the exposed areas and sends a WM_PAINT message to the window.

An application can require that the system invalidate an entire window every time the window moves or changes size. To do this, the application sets the CS_SIZEREDRAW class style in the corresponding window class. Typically, this class style is selected for use in an application that uses a window's current size and position to determine how to draw the window. For example, a clock application always would draw the face of the clock so that it filled the window exactly.

An application also can explicitly specify which parts of the window to preserve during a move or size change. Before any change is made, the system sends a WM_CALCVALIDRECTS message to windows that do not have the style CS_SIZEREDRAW. This enables the window procedure to specify what part of the window to save and where to align it after the move or size change.

Changing the Z-Order of Windows

An application can move a window to the top or bottom of the z-order by passing the SWP_ZORDER constant to the WinSetWindowPos function. An application specifies where to move the window by specifying the HWND_TOP or HWND_BOTTOM constants.

The code fragment in the following figure uses WinSetWindowPos to change the z-order of a window.

```
HWND hwndParent;
HWND hwndNext;
HENUM henum;

WinSetWindowPos(
    hwndNext,          /* Next window to move */
    HWND_TOP,          /* Put window on top */
    0, 0, 0, 0,        /* Not used for z-order */
    SWP_ZORDER);       /* Change z-order */
```

An application also can specify the window that the given window is to move behind. In this case, the application specifies the window handle instead of the HWND_TOP or HWND_BOTTOM constant.

```
HWND hwndParent;
HWND hwndNext;
HWND hwndExchange;
HENUM henum;

henum = WinBeginEnumWindows(hwndParent);

hwndExchange = WinGetNextWindow(henum);

/* hwndNext has top window;
   hwndExchange has window under the top. */

WinSetWindowPos(
    hwndNext,          /* Next window to move */
    hwndExchange,       /* Put lower window on top */
    0, 0, 0, 0,        /* Not used for z-order */
    SWP_ZORDER);       /* Change z-order */

WinEndEnumWindows(henum);
```

Showing or Hiding a Window

An application can show or hide a window by using the `WinShowWindow` function. This function changes the `WS_VISIBLE` style of a window to the specified setting. An application can also use the `WinIsWindowVisible` function to check the visibility of a window. This function returns `TRUE` if the window is visible.

Maximizing, Minimizing, and Restoring a Frame Window

An application can maximize, minimize, or restore a frame window by using the `WinSetWindowPos` function and specifying the constant `SWP_MAXIMIZE`, `SWP_MINIMIZE`, or `SWP_RESTORE`. Only a frame window can maximize and minimize by default. For any other window, an application must provide support for these actions in the corresponding window procedure.

The following figure shows how to maximize a frame window.

```
SWP swpCurrent;
HWND hwndFrame;

WinQueryWindowPos(hwndFrame, &swpCurrent);
WinSetWindowPos(
    hwndFrame,          /* Window handle          */
    NULL,               /* Not used to maximize   */
    swpCurrent.x,
    swpCurrent.y,       /* Stored for restoring window */
    swpCurrent.cx,
    swpCurrent.cy,      /* Stored for restoring window */
    SWP_MAXIMIZE | SWP_SIZE | SWP_MOVE); /* Maximize */
```

Destroying a Window

An application can destroy a window by using the `WinDestroyWindow` function. The following figure shows how to create and then destroy a control window:

```
HWND hwndCtrl;
HWND hwndParent;

hwndCtrl = WinCreateWindow(hwndParent, WC_BUTTON, ...);

WinDestroyWindow(hwndCtrl);
```

Window Classes

A *window class* determines which styles and which window procedure are given to a window when it is created. This chapter explains how a PM application creates and uses window classes.

About Window Classes

Every window is a member of a window class. An application must specify a window class when it creates a window. Each window class has an associated *window procedure* that is used by all windows of the same class. The window procedure handles messages for all windows of that class and, therefore, controls the behavior and appearance of the window.

A window class must be *registered* before an application can create a window of that class. Registering a window class associates a window procedure and class styles with a class name. When an application specifies the class name in a window-creation function such as `WinCreateWindow`, the system creates a window that uses the window procedure and styles associated with the class name.

An application can register private classes or use preregistered public window classes.

Private Window Classes

A *private window class* is any class registered within an application. An application registers a private class by calling the `WinRegisterClass` function. A private class cannot be shared with other applications. When an application terminates, the system removes any data associated with the application's private window classes.

An application can register a private class anytime but, typically, does so as part of application initialization. To register a private class during application initialization, the application also must call `WinInitialize` and, usually, `WinCreateMsgQueue` before class registration.

An application cannot de-register a private window class; it remains registered and available until the application terminates.

When an application registers a private window class, it must supply the following information:

- Class name
 - Class styles
 - Window procedure
 - Window data size
-

Class Name

The *class name* identifies the window class. The application uses this name in the window-creation functions to specify the class of the window being created. The class name can be a character string or an atom, and it must be unique within the application. The system checks as to whether a public class or a class already registered by the application has the same name. If the class name is not unique to that application, the system returns an error.

Class Styles

Each window class has one or more values, called *class styles*, that tell the system which initial window styles to give a window created with that class. An application sets the class styles for a private window class when it registers the class. Once a class is registered, the application cannot change the styles.

An application can specify one or more of the following class styles in the `WinRegisterClass` function, combining them as necessary by using the bitwise OR operator:

Style Name	Description
------------	-------------

<code>CS_CLIPCHILDREN</code>	Prevents a window from painting over its child windows, but increases the time necessary to calculate the visible region. This style usually is not necessary, because if the parent and child windows overlap and are both invalidated, the operating system draws the parent window before drawing the child window. If the child window is invalidated independently of the parent window, the system redraws only the child window. If the update region of the parent window does not intersect the child window, drawing the parent window causes the child window to be redrawn. This style is useful to prevent a child window containing a complex graphic from being redrawn unnecessarily.
<code>CS_CLIPSIBLINGS</code>	Prevents a window from painting over its sibling windows. This style protects sibling windows but increases the time necessary to calculate the visible region. This style is appropriate for windows that overlap and have the same parent window.
<code>CS_FRAME</code>	Identifies the window as a frame window.
<code>CS_HITTEST</code>	Directs the operating system to send <code>WM_HITTEST</code> messages to the window whenever the mouse pointer moves in the window.
<code>CS_MOVENOTIFY</code>	Directs the system to send <code>WM_MOVE</code> messages to the window whenever the user moves the window.
<code>CS_PARENTCLIP</code>	Extends a window's visible region to include that of its parent window. This style simplifies the calculation of the child window's visible region but, potentially, is dangerous, because the parent window's visible region is usually larger than the child window.
<code>CS_SAVEBITS</code>	Saves the screen area under a window as a bit map. When the user hides or moves the window, the system restores the image by copying the bits; there is no need to add the area to the uncovered window's update region. This style can improve system performance, but also can consume a great deal of memory. It is recommended only for transient windows such as menus and dialog windows- <i>not</i> for main application windows.
<code>CS_SIZEREDRAW</code>	Causes the window to receive a <code>WM_PAINT</code> message and be completely invalidated whenever the window is resized, even if it is made smaller. (Typically, only the uncovered area of a window is invalidated when a window is resized.) This class style is useful when an application scales graphics to fill the window.
<code>CS_SYNCPAINT</code>	Causes the window to receive <code>WM_PAINT</code> messages immediately after a part of the window becomes invalid. Without this style, the window receives <code>WM_PAINT</code> messages only if no other message is waiting to be processed.

Window Procedure

The window procedure for a window class processes all messages sent or posted to all windows of that class. It is the chief component of

the window class because it controls the appearance and behavior of each window created with the class. Window procedures are shared by all windows of a class, so an application must ensure that no conflicts arise when two windows of the same class attempt to access the same global data. In other words, the window procedure must protect global data and other shared resources.

Window Data Size

The system creates a window data structure for each window, which includes extra space that an application can use to store additional data about a window. An application specifies the number of extra bytes to allocate in the WinRegisterClass function. All windows of the same class have the same amount of window data space.

An application can store window data in a window's data structure by using the WinSetWindowUShort and WinSetWindowULong functions. It can retrieve data by using the WinQueryWindowUShort and WinQueryWindowULong functions.

Custom Window Styles

An application that registers a window class also can support its own set of styles for windows of that class. Standard window styles-for example, WS_VISIBLE and WS_SYNCPAINT-still apply to these windows. A window style is a 32-bit integer, and only the high 16 bits are used for the standard window styles; an application can use the low 16 bits for custom styles specific to a window class.

The operating system has unique window styles for all preregistered window classes. Styles such as FS_BORDER and BS_PUSHBUTTON are processed by the window procedure for the corresponding class. This means that an application can build the support for its own window styles into the window procedure for its private class. A window style designed for one window class will not work with another window class.

Public Window Classes

Public window classes are registered during system initialization. Their window procedures are in dynamic link libraries. Therefore, to use a public window class, an application need not register it. Nor does the application need to import the window procedure for a public window class because the system resolves references to the window procedure.

An application cannot use a public window class name when it registers a private window class.

System-Defined Public Window Classes

The system provides a number of public window classes that support menus, frame windows, control windows, and dialog windows. An application can create a window of a system-defined public window class by specifying one of the following class name constants in a call to WinCreateWindow:

Class Name	Description
WC_BUTTON	Consists of buttons and boxes the user can select by clicking the pointing device or using the keyboard.
WC_COMBOBOX	Creates a combination-box control, which combines a list-box control and an entry-field control. It enables the user to enter data either by typing in the entry field or by choosing from the list in the list box.
WC_CONTAINER	Creates a control in which the user can group

objects in a logical manner. A container can display those objects in various formats or views. The container control supports drag and drop so the user can place information in a container by simply dragging and dropping.

WC_ENTRYFIELD	Consists of a single line of text that the user can edit.
WC_FRAME	A composite window class that can contain child windows of many of the other window classes.
WC_LISTBOX	Presents a list of text items from which the user can make selections.
WC_MENU	Presents a list of items that can be displayed horizontally as menu bars, or vertically as pull-down menus. Usually menus are used to provide a command interface to applications.
WC_NOTEBOOK	Creates a control for the user that is displayed as a number of pages. The top page is visible, and the others are hidden, with their presence being indicated by a visible edge on each of the back pages.
WC_SCROLLBAR	Consists of window scroll bars that let the user scroll the contents of the associated window.
WC_SLIDER	Creates a control that is usable for producing approximate (analog) values or properties. Scroll bars were used for this function in the past, but the slider provides a more flexible method of achieving the same result, with less programming effort.
WC_SPINBUTTON	Creates a control that presents itself to the user as a scrollable ring of choices, giving the user quick access to the data. The user is presented only one item at a time, so the spin button should be used with data that is intuitively related.
WC_STATIC	Simple display items that do not respond to keyboard or pointing device events.
WC_TITLEBAR	Displays the window title or caption and lets the user move the window's owner.
WC_VALUESET	Creates a control similar in function to radio buttons but provides additional flexibility to display graphical, textual, and numeric formats. The values set with this control are mutually exclusive.

Each system-defined public window class has a corresponding set of window styles that an application can use to customize a window of that class. For example, a window created with the WC_BUTTON class has styles that include BS_PUSHBUTTON and BS_CHECKBOX. Window styles enable you to customize aspects of a window's behavior and appearance. The application specifies the window styles in the WinCreateWindow function.

Custom Public Window Classes

An application can create a custom public window class, but it must do so during system initialization. Only the shell can register a public window class, and it can do so only when the system starts. Registering a public window class requires a special load entry in the os2.ini file. That entry instructs the shell to load a dynamic link library whose initialization routine registers the window class. Custom public window classes must be registered using WinRegisterClass and must have the class style CS_PUBLIC. If a custom public window class registered this way has the same name as an existing public window class, the custom class replaces the original class.

If a dynamic link library replaces an existing public window class, the library can save the address of the original window procedure and use the address to subclass the original window class. The dynamic link library retrieves the original window procedure address using the `WinQueryClassInfo` function. The custom window procedure then passes unprocessed messages to the original window procedure instead of calling `WinDefWindowProc`.

When subclassing a public window class, the custom public window procedure must not make the window data size smaller than the original window data size, because all public window classes that the operating system defines use 4 extra bytes for storing a pointer to custom window data. This size is guaranteed only for public window classes defined by the operating system dynamic link libraries.

Class Data

An application can examine public window class data by using the `WinQueryClassInfo` and `WinQueryClassName` functions. An application retrieves the name of the class for a given window by using the `WinQueryClassName` function. If the window is one of the preregistered public window classes, the name returned is in the form `#nnnnnn`, where `nnnnnn` is up to 5 digits, representing the value of the window class constant. Using this window class name, the application can call `WinQueryClassInfo` to retrieve the window class data. `WinQueryClassInfo` copies the class style, window procedure address, and window data size to a `CLASSINFO` data structure.

Using Window Classes

This section explains how to perform the following tasks:

- Register a private window class
- Register an imported window procedure

Registering a Private Window Class

An application can register a private window class at any time by using the `WinRegisterClass` function. You must define the window procedure in the application, choose a unique name, and set the window styles for the class. The following code fragment shows how to register the window class name `"MyPrivateClass"`:

```
MRESULT EXPENTRY ClientWndProc(HWND hwnd,ULONG msg,MPARAM mp1, MPARAM mp2);

HAB hab;

WinRegisterClass(hab,          /* Anchor block handle */
    "MyPrivateClass",         /* Name of class being registered */
    ClientWndProc,            /* Window procedure for class */
    CS_SIZEREDRAW |           /* Class style */
    CS_HITTEST,               /* Class style */
    0);                       /* Extra bytes to reserve */
```

Window Procedures

Windows have an associated *window procedure*-a function that processes all messages sent or posted to a window. Every aspect of a

window's appearance and behavior depends on the window procedure's response to the messages. This chapter explains how window procedures function, in general, and describes the default window procedure.

About Window Procedures

Every window belongs to a window class that determines which window procedure a particular window uses to process its messages. All windows of the same class use the same window procedure. For example, the operating system defines a window procedure for the frame window class (WC_FRAME), and all frame windows use that window procedure.

An application typically defines at least one new window class and an associated window procedure. Then, the application can create many windows of that class, all of which use the same window procedure. This means that the same piece of code can be called from several sources simultaneously; therefore, you must be careful when modifying shared resources from a window procedure.

Dialog procedures have the same structure and function as window procedures. The primary difference between a dialog procedure and a window procedure is the absence of a client window in the dialog procedure; that is, the controls in a dialog procedure are the immediate child windows of the frame, whereas the controls in a normal window are the *grandchildren* of the frame. This makes significant differences in the code between the two; for example, WinSendDlgItemMsg does not work from a client window if you pass the client window handle as the first parameter.

Structure of a Window Procedure

A window procedure is a function that takes 4 arguments and returns a 32-bit pointer. The arguments of a window procedure consist of a window handle, a ULONG message identifier, and two arguments, called *message parameters*, that are declared with the MPARAM data type. The system defines an MPARAM as a 32-bit pointer to a VOID data type (a generic pointer). The message parameters actually might contain any of the standard data types. The message parameters are interpreted differently, depending on the value of the message identifier. The operating system includes several macros that enable the application to cast the information from the MPARAM values into the actual data type. SHORT1FROMMP, for example, extracts a 16-bit value from a 32-bit MPARAM. The window-procedure arguments are described in the following table:

Argument	Description
<code>hwnd</code>	Handle of the window receiving the message.
<code>msg</code>	Message identifier. The message will correspond to one of the predefined constants (for example, WM_CREATE) defined in the system include files or be an application-defined message identifier. The value of an application-defined message identifier must be greater than the value of WM_USER, and less than or equal to 0xffff.
<code>mp1, mp2</code>	Message parameters. Their interpretation depends on the particular message.

The return value of a window procedure is defined as an HRESULT data type. The interpretation of the return value depends on the particular message. Consult the description of each message to determine the appropriate return value.

Default Window Procedure

All windows in the system share certain fundamental behavior, defined in the default window-procedure function, WinDefWindowProc. The default window procedure provides the minimal functionality for a window. An application-defined window procedure should pass any messages it does not process to WinDefWindowProc for default processing.

Window-Procedure Subclassing

Subclassing enables an application to intercept and process messages sent or posted to a window before that window has a chance to process them. Subclassing most often is used to add functionality to a particular window or to alter a window's default behavior.

An application subclasses a window by using the `WinSubclassWindow` function to replace the window's original window procedure with an application-defined window procedure. Thereafter, the new window procedure processes any messages that are sent or posted to the window. If the new window procedure does not process a particular message, it must pass the message to the original window procedure, *not* to `WinDefWindowProc`, for default processing.

Using Window Procedures

This section explains how to:

- Design a window procedure
- Associate a window procedure with a window class
- Subclass a window

Designing a Window Procedure

The following code fragment shows the structure of a typical window procedure and how to use the message argument in a switch statement, with individual messages handled by separate case statements. Notice that each case returns a specific value for each message. For messages that it does not handle itself, the window procedure calls `WinDefWindowProc`.

```
MRESULT ClientWndProc(
    HWND hwnd,
    ULONG msg,
    MPARAM mp1,
    MPARAM mp2)
{
    /* Define local variables here, if required. */
    switch (msg) {
        case WM_CREATE:

            /* Initialize private window data.          */
            return (MRESULT) FALSE;

        case WM_PAINT:

            /* Paint the window.                          */
            return 0;

        case WM_DESTROY:

            /* Clean up private window data.              */
            return 0;

        default:
            break;
    }
    return WinDefWindowProc (hwnd, msg, mp1, mp2);
}
```

A dialog window procedure does not receive the `WM_CREATE` message; however, it does receive a `WM_INITDLG` message when all of its control windows have been created. At the very least, a window procedure should handle the `WM_PAINT` message to draw itself. Typically, it should handle mouse and keyboard messages as well. Consult the descriptions of individual messages to determine whether your window

procedure should handle them.

An application can call WinDefWindowProc as part of the processing of a message. In such a case, the application can modify the message parameters before passing the message to WinDefWindowProc or can continue with the default processing after performing its own operations.

Associating a Window Procedure with a Window Class

To associate a window procedure with a window class, an application must pass a pointer to that window procedure to the WinRegisterClass function. Once an application has registered the window procedure, the procedure automatically is associated with each new window created with that class.

The following code fragment shows how to associate the window procedure in the previous example with a window class:

```
HAB hab;
CHAR szClientClass[] = "My Window Class";

WinRegisterClass(hab,      /* Anchor-block handle */
                 szClientClass, /* Class name */
                 ClientWndProc, /* Pointer to procedure */
                 CS_SIZEREDRAW, /* Class style */
                 0);        /* Window data */
```

Subclassing a Window

To subclass a window, an application calls the WinSubclassWindow function, specifying the handle of the window to subclass and a pointer to the new window procedure. The WinSubclassWindow function returns a pointer to the original window procedure; the application can use this pointer to pass unprocessed messages to the original procedure. The following code fragment subclasses a push button control window. The new window procedure generates a beep whenever the user clicks the push button.

```
PFNWP pfnPushBtn;
CHAR szCancel[] = "Cancel";
HWND hwndClient;
HWND hwndPushBtn;
.
.
.

/* Create a push button control. */
hwndPushBtn = WinCreateWindow(
    hwndClient,      /* Parent-window handle */
    WC_BUTTON,      /* Window class */
    szCancel,        /* Window text */
    WS_VISIBLE |    /* Window style */
    WS_SYNCPAINT |  /* Window style */
    BS_PUSHBUTTON,   /* Button style */
    50, 50,          /* Physical position */
    70, 30,          /* Width and height */
    hwndClient,      /* Owner-window handle */
    HWND_TOP,        /* Z-order position */
    1,               /* Window identifier */
    NULL,            /* No control data */
    NULL);           /* No presentation parameters */

/* Subclass the push button control. */
pfnPushBtn = WinSubclassWindow(hwndPushBtn,
```



```

        SubclassPushBtnProc);
        .
        .
        .
    }
    /* This procedure subclasses the push button. */
    MRESULT EXPENTRY SubclassPushBtnProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
    {
        switch (msg) {

            /* Beep when the user clicks the push button. */
            case WM_BUTTON1DOWN:
                DosBeep(1000, 250);
                break;

            default:
                break;
        }

        /* Pass all messages to the original window procedure. */
        return (MRESULT) pfnPushBtn(hwnd, msg, mp1, mp2);
    }

```

Window Timers

A *window timer* enables an application to post timer messages at specified intervals. This chapter describes how to use window timers in PM applications.

About Window Timers

A window timer causes the system to post WM_TIMER messages to a message queue at specified time intervals called *timeout values*. A timeout value is expressed in milliseconds.

An application starts the timer for a given window, specifying the timeout value. The system counts down approximately that number of milliseconds and posts a WM_TIMER message to the message queue for the corresponding window. The system repeats the countdown-post cycle continuously until the application stops the timer.

The timeout value can be any value in the range from 0 through 4,294,967,295 (full magnitude of *ULONG*) for OS/2 Version 3; for previous versions, the maximum value is 65535. However, the operating system cannot guarantee that all values are accurate. The actual timeout depends on how often the application retrieves messages from the queue and the system clock rate. In many computers, the operating system clock ticks about every 50 milliseconds, but this can vary widely from computer to computer. In general, a timer message cannot be posted more frequently than every system clock tick. To make the system post a timer message as often as possible, an application can set the timeout value to 0.

An application starts a timer by using the WinStartTimer function. If a window handle is given, the timer is created for that window. In such case, the WinDispatchMsg function dispatches the WM_TIMER message to the given window when the message is retrieved from the message queue. If a NULL window handle is given, it is up to the application to check the message queue for WM_TIMER messages and dispatch them to the appropriate window.

A new timer starts counting down as soon as it is created. An application can reset or change a timer's timeout value in subsequent calls to the WinStartTimer function. To stop a timer, an application can use the WinStopTimer function.

The system contains a limited number of timers that must be shared among all PM applications; each application should use as few timers as possible. An application can determine how many timers currently are available by checking the SV_CTIMERS system value.

Every timer has a unique timer identifier. An application can request that a timer be created with a particular identifier or have the system choose a unique value. When a WM_TIMER message is received, the timer identifier is contained in the first message parameter. Timer identifiers enable an application to determine the source of the WM_TIMER message. Three timer identifiers are reserved by and for the system and cannot be used by applications; these system timer identifiers and their symbolic constants are shown in the following table:

Value	Meaning
<code>TID_CURSOR</code>	Identifies the timer that controls cursor blinking. Its timeout value is stored in the <code>os2.ini</code> file under the <code>CursorBlinkRate</code> keyname in the <code>PM_ControlPanel</code> section.
<code>TID_FLASHWINDOW</code>	Identifies the window-flashing timer.
<code>TID_SCROLL</code>	Identifies the scroll-bar repetition timer that controls scroll-bar response when the mouse button or a key is held down. Its timeout value is specified by the system value <code>SV_SCROLLRATE</code> .

WM_TIMER messages, like WM_PAINT and semaphore messages, are not actually posted to a message queue. Instead, when the time elapses, the system sets a record in the queue indicating which timer message was posted. The system builds the WM_TIMER message when the application retrieves the message from the queue.

Although a timer message may be in the queue, if there are any messages with higher priority in the queue, the application retrieves those messages first. If the time elapses again before the message is retrieved, the system does not create a separate record for this timer, meaning that the application should not depend on the timer messages being processed at precise intervals. To check the accuracy of the message, an application can retrieve the actual system time by using the `WinGetCurentTime` function. Comparing the actual time with the time of the previous timer message is useful in determining what action to take for the timer.

Using Window Timers

There are two methods of using window timers. In the first method, you start the timer by using the `WinStartTimer` function, supplying the window handle and timer identifier. The function associates the timer with the specified window. The following code fragment starts two timers: the first timer is set for every half second (500 milliseconds); the second, for every two seconds (2000 milliseconds).

```
WinStartTimer(hab, /* Anchor-block handle */
             hwnd, /* Window handle */
             ID_TIMER1, /* Timer identifier */
             500); /* 500 milliseconds */

WinStartTimer(hab, /* Anchor-block handle */
             hwnd, /* Window handle */
             ID_TIMER2, /* Timer identifier */
             2000); /* 2000 milliseconds */
```

Once these timers are started, the `WinDispatchMsg` function dispatches WM_TIMER messages to the appropriate window. To process these messages, add a WM_TIMER case to the window procedure for the given window. By checking the first parameter of the WM_TIMER message, you can identify a particular timer, then carry out the actions related to it. The following code fragment shows how to process WM_TIMER messages:

```
case WM_TIMER:
    switch (SHORT1FROMMP(mp1)) { /* Obtains timer identifier */
        case ID_TIMER1:
            . /* Carry out timer-related tasks. */
            .
            return 0;

        case ID_TIMER2:
```

```

        . /* Carry out timer-related tasks.          */
        .
        . return 0;
    }

```

In the second method of using a timer, you specify NULL as the *hwnd* parameter of the WinStartTimer call. The system starts a timer that has no associated window and assigns an arbitrary timer identifier. The following code fragment starts two window timers using this method:

```

ULONG idTimer1, idTimer2;

idTimer1 = WinStartTimer(hab, (HWND) NULL, 0, 500);
idTimer2 = WinStartTimer(hab, (HWND) NULL, 0, 2000);

```

These timers have no associated window, so the application must check the message queue for WM_TIMER messages and dispatch them to the appropriate window procedure. The following code fragment shows a message loop that handles the window timers:

```

HWND hwndTimerHandler; /* Handle of window for timer messages */
QMSG qmsg;              /* Queue-message structure             */

while (WinGetMsg(hab, &qmsg, (HWND) NULL, 0, 0)) {
    if (qmsg.msg == WM_TIMER)
        qmsg.hwnd = hwndTimerHandler;
    WinDispatchMsg(hab, &qmsg);
}

```

You can use the WinStopTimer function at any time to stop a timer. The following code fragment demonstrates how to stop a timer:

```

WinStopTimer(hab, hwnd, ID_TIMER1); /* Stops first timer */

```

Appendices

This section contains several topics related to Presentation Manager programming.

Bitmap Formats

There are four standard bitmap formats. All device drivers must be able to translate between any of these formats and their own internal formats. The standard formats are:

Bitcount	Planes
1	1
4	1
8	1
24	1

These formats are chosen because they are identical or similar to all formats commonly used by raster devices. Only single-plane formats are standard, but it is very easy to convert these to any multiple-plane format used internally by a device.

Bitmap Data

The pel data is stored in the bitmap in the order that the coordinates appear on a display screen. That is, the pel in the lower-left corner is the first in the bitmap. Pels are scanned to the right, and upward, from that position. The bits of the first pel are stored, beginning with the most significant bits of the first byte. The data for pels in each scan line is packed together tightly, but all scan lines are padded at the end, so that each one begins on a [ULONG](#) boundary.

Bitmap Information Tables

Each standard-format bitmap must be accompanied by a bitmap information table. Because the standard-format bitmaps are intended to be traded between devices, the color indexes in the bitmap are meaningless without more information; for a description of this structure, see [BITMAPINFO2](#).

Some functions use a structure that is similar to [BITMAPINFO2](#) but does not have the color table array; for a description of this structure, see [BITMAPINFOHEADER2](#). Wherever [BITMAPINFO2](#) is shown, [BITMAPINFO](#) is also allowed. Similarly, wherever [BITMAPINFOHEADER2](#) is shown, [BITMAPINFOHEADER](#) is also allowed.

Bitmap Example

To make the ordering of all the bytes clear, consider this simple example of a 5-by-3 array of colored pels:

```
Red   Green Blue  Red   Green
Blue  Red   Green Blue  Red
Green Blue  Red   Green Blue

ULONG ExampleBitmap[] {
    0x23,0x12,0x30,0x00          /* bottom line */
    0x31,0x23,0x10,0x00          /* middle line */
    0x12,0x31,0x20,0x00          /* top line    */
};

#define BLACK 0x00000000L
#define RED   0x00FF0000L
#define GREEN 0x0000FF00L
#define BLUE  0x000000FFL

struct BitmapInfoTable ExampleInfo = {
    5,          /* width      */
    3,          /* height     */
    1,          /* planes     */
    4,          /* bitcount   */
    BLACK,RED, GREEN,BLUE, /* color table */
    BLACK,BLACK,BLACK,BLACK,
    BLACK,BLACK,BLACK,BLACK,
    BLACK,BLACK,BLACK,BLACK
};
```

Bitmap File Format

The operating system uses the same file format for bitmaps, icons, and pointers in resource files. In the following description, "bitmap" refers to bitmaps, icons, and pointers unless otherwise specified.

Two formats are supported. In the first, a single-size version of the bitmap is defined. This is used whatever the target device.

The second format allows multiple versions of the bitmap to be defined, including one or more device-independent versions, and a number of device-dependent versions, each intended for use with a particular device.

In the case of icons and pointers, when more than one version of the bitmap exists, the preferred version is one that matches the device size of the icon or pointer; otherwise, the device-independent version is used to scale a bitmap to the required size.

The operating system provides pointers that match the requirements of the display device in use, typically pointers are 32x32 pels, one bit per plane.

Icons provided with the operating system are designed to match the requirements of the most common display devices. The following versions of each icon are included in each file:

- 32x32 4 bpp (16 color)
- 40x40 4 bpp (16 color)
- 32x32 1 bpp (black and white)
- 20x20 1 bpp (black and white)
- 16x16 1 bpp (black and white)

The 32x32 versions are designed for VGA displays and for device-independent use.

The 40x40 version is for 8514/A and XGA displays.

The 20x20 and 16x16 are half-size icons designed for use as mini-icons.

For general bitmaps, which may be of arbitrary size, the preferred version is one matching the requested bitmap size; otherwise one matching the display size is selected. If neither is available, the device-independent version is used from which to scale a bitmap.

For both formats, the definition consists of two sections. The first section contains general information about the type, dimensions, and other attributes of the resource. The second section contains data describing the pels that make up the bitmap(s), and is in the format specified in [Bitmap Data](#).

In the multiple-version format, the first section contains an array of [BITMAPARRAYFILEHEADER](#) or [BITMAPARRAYFILEHEADER2](#) structures.

The device-independent version must be the first [BITMAPARRAYFILEHEADER](#) or [BITMAPARRAYFILEHEADER2](#) defined.

In the single-size format, the [BITMAPARRAYFILEHEADER](#) or [BITMAPARRAYFILEHEADER2](#) structure is not present. The definition consists of one or two [BITMAPFILEHEADER](#) or [BITMAPFILEHEADER2](#) structures.

For icons and pointers, the *cy* field in *bmp* is actually twice the pel height of the image that appears on the screen. This is because these types actually contain two full bitmap pel definitions. The first bitmap definition is the XOR mask, which contains invert information (0 = no invert, 1 = invert) for the pointer or icon. The second is the AND mask, which determines whether the pointer or the screen is shown (0 = black/white, 1 = screen/inverse screen).

For color icons or pointers, there are two bitmaps involved: one that is black and white and consists of an AND and an XOR mask, and one that is color that defines the color content.

The *cy* field in the [BITMAPINFOHEADER2](#) structure for the color bitmap must be the real height, that is, half the value specified for the black and white bitmap. The *cx* must be the same.

The following table shows how these two bitmaps are used for a color icon or pointer:

XOR	AND	COLOR	
1	1	x	Invert screen
0	0	x	Use color x
0	1	x	Transparency
1	0	x	Use color x

For color icons or pointers, two [BITMAPFILEHEADER](#) or [BITMAPFILEHEADER2](#) structures are therefore required:

```
BITMAPFILEHEADER2    with usType BFT_COLORICON or BFT_COLORPOINTER
  BITMAPINFOHEADER2 (part of BITMAPFILEHEADER2)
  Color table
```

```

BITMAPFILEHEADER2    with same usType
    BITMAPINFOHEADER2 (part of BITMAPFILEHEADER2)
        Color table
**
bits for one bitmap
**
**
bits for other bitmap
**

```

The *usType* for the first [BITMAPFILEHEADER2](#) is either BFT_COLORICON or BFT_COLORPOINTER. This means that a second [BITMAPFILEHEADER2](#) is present as part of the definition of a color icon or pointer. The first [BITMAPFILEHEADER2](#) structure contains the information for the black and white AND and XOR masks, while the second [BITMAPFILEHEADER2](#) structure contains the information for the color part of the pointer or icon.

[BITMAPFILEHEADER](#) and [BITMAPINFOHEADER](#) can occur in place of [BITMAPFILEHEADER2](#) and [BITMAPINFOHEADER2](#) in this example.

For the multiple version format, the file is as follows:

```

BITMAPARRAYFILEHEADER2  for device-independent version
    BITMAPFILEHEADER2    (part of BITMAPARRAYFILEHEADER2)
        BITMAPINFOHEADER2 (part of BITMAPFILEHEADER2)
            Color table

    BITMAPFILEHEADER2    )
        BITMAPINFOHEADER2 ) only if this is a color icon or pointer
            Color table   )

BITMAPARRAYFILEHEADER2  for first device-dependent version
    BITMAPFILEHEADER2    (part of BITMAPARRAYFILEHEADER2)
        BITMAPINFOHEADER2 (part of BITMAPFILEHEADER2)
            Color table

    BITMAPFILEHEADER2    )
        BITMAPINFOHEADER2 ) only if this is a color icon or pointer
            Color table   )

```

Further [BITMAPARRAYFILEHEADER2](#) groups occur here as required for additional device-dependent versions

```

**
bits for one bitmap
**
**
bits for next bitmap
**

```

And so on for as many bitmaps as necessary.

As before, [BITMAPARRAYFILEHEADER](#), [BITMAPFILEHEADER](#), and [BITMAPINFOHEADER](#), can occur in place of [BITMAPARRAYFILEHEADER2](#), [BITMAPFILEHEADER2](#), and [BITMAPINFOHEADER2](#)

Code Pages

The initialization file contains country information relating to date, time, and numeric formats. It does not contain code-page information; this is obtained from the CONFIG.SYS file.

Applications start with the default code page. The default code page is set when the operating system is installed. It can be changed subsequently either by reinstalling the operating system or by editing the COUNTRY statement in the CONFIG.SYS file.

A GPI presentation space inherits the code page of the process that created it. The code page changes only when the process calls `GpiSetCp`

See the printed version of the *Presentation Manager Programming Reference* for the ASCII and EBCDIC versions of the code pages.

Windowed PM Applications

Windowed PM applications allow the code-page calls to use any of the supported ASCII code pages. These are:

	Char. Set	Code Page
Canadian-French	993	863
Desktop Publishing	1146	1004
Iceland	991	861
Latin 1 Multilingual	980	850
Latin 2 Multilingual	982	852
Nordic	995	865
Portuguese	990	860
Turkey	987	857
U.S. (IBM PC)	919	437

Code page 1004 is compatible with Microsoft** Windows**.

The following EBCDIC code pages, based on character set 697, are also available for output:

	Char. Set	Code Page
Austrian/German	697	273
Belgian	697	500
Brazil	697	037
Czechoslovakia	959	870
Danish/Norwegian	697	277
Finnish/Swedish	697	278
French	697	297
Hungary	959	870
Iceland	697	871
International	697	500
Italian	697	280
Poland	959	870
Portuguese	697	037
Spanish	697	284
Turkey	1152	1026
U.K.-English	697	285
U.S.-English	697	037
Yugoslavia	959	870

Note: Code pages 274 (Belgian) and 282 (Portuguese) can be used to provide access to old data.

The operating system provides the following additional code-page setting and query calls for the supported ASCII and EBCDIC code pages. These calls work independently of the CONFIG.SYS file.

GpiSetCp	Sets the code page for GPI.
GpiQueryCp	Queries the code page for GPI.
GpiCreateLogFont	Creates fonts in a code page.
WinSetCp	Sets the code page for a message queue.
WinQueryCp	Queries the code page for a message queue.

WinQueryCpList creates a list of code pages supported by the operating system.

Text entered in a dialog box is supplied to the application in the code page of the queue ("queue code page"). If possible, the code page of a resource (for example, a menu or dialog box) should match the code page of the queue. In general, code page 850 is the best choice for both an application and its resources.

Applications should be able to process data from a variety of sources. Because code page 850 contains most of the characters in other supported code pages, this is usually the best choice for the queue code page.

OS/2 Code Page Options for PM Applications

Application	CONFIG.SYS
DosSetProcessCp (see note 1) Set code page for this process (keyboard/display not changed).	contains the default code page set by CODEPAGE=
WinQueryCpList (see note 2) Query list of supported code pages.	
WinSetCp, WinQueryCp (see note 1) Set or query code page for translating incoming messages (keystrokes).	`Keyboard Message queue
GpiSetCp, GpiQueryCp (see note 2) Set or query default GPI code page.	
GpiCreateLogFont (see note 2) Create font in a code page.	Display
WinCpTranslateChar (see note 2) WinCpTranslateString (see note 2) Convert character or string from one code page to another.	Disk LAN or host

- Note 1: Either of the two ASCII code pages specified in CONFIG.SYS. Code page 1004 is also supported.
- Note 2: Any supported ASCII or EBCDIC code page as reported by WinQueryCpList. Code page 1004 is also supported.

OS/2 Font Support for Multiple Code Pages

The operating system supports multiple code pages for text input and output. A single font resource is used to support all the code pages. This section describes the font resource format.

Font Code-Page Functions

Many of the characters required by each code page are common; for example, the first 128 characters of all the ASCII code pages are identical. This set of characters is called the Universal Glyph List (UGL). A code page is simply a set of pointers into the UGL.

As the characters in every font are in the same order, only one set of code-page translation tables is necessary.

Note: The fonts of Microsoft Windows support only code page 1004.

Font Layout

The following table lists the full character set in the order in which the characters occur in the multi-code-page font. Characters are listed in order of their universal glyph list (UGL) number; the graphic character global identifier (GCGID) and a description of each character are also given.

UGL	GCGID	Description
1	SS000000	Smiling face
2	SS010000	Smiling face, reverse image
3	SS020000	Heart suit symbol
4	SS030000	Diamond suit symbol
5	SS040000	Club suit symbol
6	SS050000	Spade suit symbol
7	SM570000	Bullet
8	SM570001	Bullet, reverse image
9	SM750000	Open circle
10	SM750002	Open circle, reverse image
11	SM280000	Male symbol
12	SM290000	Female symbol
13	SM930000	Musical note
14	SM910000	Two musical notes
15	SM690000	Sun symbol
16	SM590000	Forward arrow indicator
17	SM630000	Back arrow indicator
18	SM760000	Up-down arrow
19	SP330000	Double exclamation point
20	SM250000	Paragraph symbol (USA)

21	SM240000	Section symbol (USA), paragraph (Europe)
22	SM700000	Solid horizontal rectangle
23	SM770000	Up-down arrow, perpendicular
24	SM320000	Up arrow
25	SM330000	Down arrow
26	SM310000	Right arrow
27	SM300000	Left arrow
28	SA420000	Right angle symbol
29	SM780000	Left-right arrow
30	SM600000	Solid triangle
31	SV040000	Solid triangle, inverted
32	SP010000	Space
33	SP020000	Exclamation point
34	SP040000	Quotation marks
35	SM010000	Number sign
36	SC030000	Dollar sign
37	SM020000	Percent sign
38	SM030000	Ampersand
39	SP050000	Apostrophe
40	SP060000	Left parenthesis
41	SP070000	Right parenthesis
42	SM040000	Asterisk
43	SA010000	Plus sign
44	SP080000	Comma
45	SP100000	Hyphen/minus sign
46	SP110000	Period/full stop
47	SP120000	Slash
48	ND100000	Zero
49	ND010000	One
50	ND020000	Two
51	ND030000	Three
52	ND040000	Four
53	ND050000	Five
54	ND060000	Six
55	ND070000	Seven
56	ND080000	Eight
57	ND090000	Nine
58	SP130000	Colon
59	SP140000	Semicolon
60	SA030000	Less than sign/greater than (arabic)

61	SA040000	Equal Sign
62	SA050000	Greater than sign/less than (arabic)
63	SP150000	Question mark
64	SM050000	At sign
65	LA020000	A capital
66	LB020000	B capital
67	LC020000	C capital
68	LD020000	D capital
69	LE020000	E capital
70	LF020000	F capital
71	LG020000	G capital
72	LH020000	H capital
73	LI020000	I capital
74	LJ020000	J capital
75	LK020000	K capital
76	LL020000	L capital
77	LM020000	M capital
78	LN020000	N capital
79	LO020000	O capital
80	LP020000	P capital
81	LQ020000	Q capital
82	LR020000	R capital
83	LS020000	S capital
84	LT020000	T capital
85	LU020000	U capital
86	LV020000	V capital
87	LW020000	W capital
88	LX020000	X capital
89	LY020000	Y capital
90	LZ020000	Z capital
91	SM060000	Left bracket
92	SM070000	Backslash
93	SM080000	Right bracket
94	SD150000	Circumflex Accent
95	SP090000	Underline, continuous underscore
96	SD130000	Grave accent
97	LA010000	a small
98	LB010000	b small
99	LC010000	c small

100	LD010000	d small
101	LE010000	e small
102	LF010000	f small
103	LG010000	g small
104	LH010000	h small
105	LI010000	i small
106	LJ010000	j small
107	LK010000	k small
108	LL010000	l small
109	LM010000	m small
110	LN010000	n small
111	LO010000	o small
112	LP010000	p small
113	LQ010000	q small
114	LR010000	r small
115	LS010000	s small
116	LT010000	t small
117	LU010000	u small
118	LV010000	v small
119	LW010000	w small
120	LX010000	x small
121	LY010000	y small
122	LZ010000	z small
123	SM110000	Left brace
124	SM130000	Vertical line, logical OR
125	SM140000	Right brace
126	SD190000	Tilde
127	SM790000	House
128	LC420000	C cedilla capital
129	LU170000	U diaeresis small
130	LE110000	E acute small
131	LA150000	A circumflex small
132	LA170000	A diaeresis small
133	LA130000	A grave small
134	LA270000	A overcircle small
135	LC410000	C cedilla small
136	LE150000	E circumflex small
137	LE170000	E diaeresis small
138	LE130000	E grave small
139	LI170000	I diaeresis small

140	LI150000	I circumflex small
141	LI130000	I grave small
142	LA180000	A diaeresis capital
143	LA280000	A overcircle capital
144	LE120000	E acute capital
145	LA510000	AE diphthong small
146	LA520000	AE diphthong capital
147	LO150000	O circumflex small
148	LO170000	O diaeresis small
149	LO130000	O grave small
150	LU150000	U circumflex small
151	LU130000	U grave small
152	LY170000	Y diaeresis small
153	LO180000	O diaeresis capital
154	LU180000	U diaeresis capital
155	LO610000	O slash small
156	SC020000	Pound sterling sign
157	LO620000	O slash capital
158	SA070000	Multiply sign
159	SC070000	Florin sign
160	LA110000	A acute small
161	LI110000	I acute small
162	LO110000	O acute small
163	LU110000	U acute small
164	LN190000	N tilde small
165	LN200000	N tilde capital
166	SM210000	Ordinal indicator, feminine
167	SM200000	Ordinal indicator, masculine
168	SP160000	Question mark, inverted
169	SM530000	Registered trademark symbol
170	SM660000	Logical NOT, end of line symbol
171	NF010000	One-half
172	NF040000	One-quarter
173	SP030000	Exclamation point, inverted
174	SP170000	Left angled quotes
175	SP180000	Right angled quotes
176	SF140000	Fill character, light
177	SF150000	Fill character, medium
178	SF160000	Fill character, heavy

179	SF110000	Center box bar vertical
180	SF090000	Right middle box side
181	LA120000	A acute capital
182	LA160000	A circumflex capital
183	LA140000	A grave capital
184	SM520000	Copyright symbol
185	SF230000	Right box side double
186	SF240000	Center box bar vertical double
187	SF250000	Upper right box corner double
188	SF260000	Lower right box corner double
189	SC040000	Cent sign
190	SC050000	Yen sign
191	SF030000	Upper right box corner
192	SF020000	Lower left box corner
193	SF070000	Middle box bottom
194	SF060000	Middle box top
195	SF080000	Left middle box side
196	SF100000	Center box bar horizontal
197	SF050000	Box intersection
198	LA190000	A tilde small
199	LA200000	A tilde capital
200	SF380000	Lower left box corner double
201	SF390000	Upper left box corner double
202	SF400000	Middle box bottom double
203	SF410000	Middle box top double
204	SF420000	Left box side double
205	SF430000	Center box bar horizontal double
206	SF440000	Box intersection double
207	SC010000	International currency symbol
208	LD630000	eth Icelandic small
209	LD620000	D stroke capital, Eth Icelandic capital
210	LE160000	E circumflex capital
211	LE180000	E diaeresis capital
212	LE140000	E grave capital
213	LI610000	I dotless small
214	LI120000	I acute capital
215	LI160000	I circumflex capital
216	LI180000	I diaeresis capital
217	SF040000	Lower right box corner
218	SF010000	Upper left box corner

219	SF610000	Solid fill character
220	SF570000	Solid fill character, bottom half
221	SM650000	Vertical line, broken
222	LI140000	I grave capital
223	SF600000	Solid fill character, top half
224	LO120000	O acute capital
225	LS610000	Sharp s small
226	LO160000	O circumflex capital
227	LO140000	O grave capital
228	LO190000	O tilde small
229	LO200000	O tilde capital
230	SM170000	Micro symbol
231	LT630000	Thorn Icelandic small
232	LT640000	Thorn Icelandic capital
233	LU120000	U acute capital
234	LU160000	U circumflex capital
235	LU140000	U grave capital
236	LY110000	y acute small
237	LY120000	Y acute capital
238	SM150000	Overline
239	SD110000	Acute accent
240	SP320000	Syllable hyphen
241	SA020000	Plus or minus sign
242	SM100000	Double underscore
243	NF050000	Three-quarters
244	SM250000	Paragraph symbol (USA)
245	SM240000	Section symbol (USA), paragraph (Europe)
246	SA060000	Divide sign
247	SD410000	Cedilla (or sedila) accent
248	SM190000	Degree symbol
249	SD170000	Diaeresis, umlaut accent
250	SD630000	Middle dot
251	ND011000	One superscript
252	ND031000	Three superscript
253	ND021000	Two superscript
254	SM470000	Solid square, histogram, square bullet
255	SP300000	Required space
256	SC060000	Peseta sign
257	SM680000	Start of line symbol

258	SF190000	Right box side double to single
259	SF200000	Right box side single to double
260	SF210000	Upper right box corner single to double
261	SF220000	Upper right box corner double to single
262	SF270000	Lower right box corner single to double
263	SF280000	Lower right box corner double to single
264	SF360000	Left box side single to double
265	SF370000	Left box side double to single
266	SF450000	Middle box bottom single to double
267	SF460000	Middle box bottom double to single
268	SF470000	Middle box top double to single
269	SF480000	Middle box top single to double
270	SF490000	Lower left box corner double to single
271	SF500000	Lower left box corner single to double
272	SF510000	Upper left box corner single to double
273	SF520000	Upper left box corner double to single
274	SF530000	Box intersection single to double
275	SF540000	Box intersection double to single
276	SF580000	Solid fill character, left half
277	SF590000	Solid fill character, right half
278	GA010000	Alpha small
279	GG020000	Gamma capital
280	GP010000	Pi small
281	GS020000	Sigma capital
282	GS010000	Sigma small
283	GT010000	Tau small
284	GF020000	Phi capital
285	GT620000	Theta capital
286	GO320000	Omega capital
287	GD010000	Delta small
288	SA450000	Infinity symbol
289	GF010000	Phi small
290	GE010000	Epsilon small
291	SA380000	Intersection, logical product
292	SA480000	Indentity symbol, almost equal
293	SA530000	Greater than or equal sign
294	SA520000	Less than or equal sign
295	SS260000	Upper integral symbol section
296	SS270000	Lower integral symbol section
297	SA700000	Nearly equals symbol

298	SA790000	Product dot
299	SA800000	Radical symbol
300	LN011000	N small superscript
301	SD310000	Macron accent
302	SD230000	Breve accent
303	SD290000	Overdot accent (over small Alpha)
304	SD270000	Overcircle accent
305	SD250000	Double acute accent
306	SD430000	Ogonek accent
307	SD210000	Caron accent
308	SP190000	Left single quote
309	SP200000	Right single quote
310	SP210000	Left double quotes
311	SP220000	Right double quotes
312	SS680000	Endash
313	SM900000	Emdash
314	SD150000	Circumflex accent
315	SD190000	Tilde accent
316	SP260000	Single quote on baseline (German lower)
317	SP230000	Left lower double quotes
318	SV520000	Ellipsis
319	SM340000	Dagger footnote indicator
320	SM350000	Double dagger footnote indicator
321	SD150100	Circumflex accent (over small alpha)
322	SM560000	Per mille symbol
323	LS220000	S caron capital
324	SP270000	French single open quote
325	LO520000	OE ligature capital
326	SD190100	Tilde accent (over small alpha)
327	SM540000	Trademark symbol
328	LS210000	s caron small
329	SP280000	French single close quote
330	LO510000	oe ligature small
331	LY180000	Y diaeresis capital
332	LG230000	g Breve Small
333	LG240000	G Breve Capital
334	LI300000	I Overdot Capital
335	LS410000	s Cedilla Small
336	LS420000	S Cedilla Capital

337	LA230000	a Breve Small
338	LA240000	A Breve Capital
339	LA430000	a Ogonek Small
340	LA440000	A Ogonek Capital
341	LC110000	c Acute Small
342	LC120000	C Acute Capital
343	LC210000	c Caron Small
344	LC220000	C Caron Capital
345	LD210000	d Caron Small
346	LD220000	D Caron Capital
347	LD610000	d Stroke Small
348	LE210000	e Caron Small
349	LE220000	E Caron Capital
350	LE430000	e Ogenek Small
351	LE440000	E Ogonek Capital
352	LL110000	l Acute Small
353	LL120000	L Acute Capital
354	LL210000	l Caron Small
355	LL220000	L Caron Capital
356	LL610000	l Stroke Small
357	LL620000	L Stroke Capital
358	LN110000	n Acute Small
359	LN120000	N Acute Capital
360	LN210000	n Caron Small
361	LN220000	N Caron Capital
362	LO250000	o Double Acute Small
363	LO260000	O Double Acute Capital
364	LR110000	r Acute Small
365	LR120000	R Acute Capital
366	LR210000	r Caron Small
367	LR220000	R Caron Capital
368	LS110000	s Acute Small
369	LS120000	S Acute Capital
370	LT210000	t Caron Small
371	LT220000	T Caron Capital
372	LT410000	t Cedilla Small
373	LT420000	T Cedilla Capital
374	LU250000	u Double Acute Small
375	LU260000	U Double Acute Capital

376	LU270000	u Overcircle Small
377	LU280000	u Overcircle Capital
378	LZ110000	z Acute Small
379	LZ120000	Z Acute Capital
380	LZ210000	z Caron Small
381	LZ220000	Z Caron Capital
382	LZ290000	z Overdot Small
383	LZ300000	Z Overdot Capital

Fonts Supplied with the OS/2 Operating System

OS/2 outline fonts and Presentation Manager bit map fonts are supplied by the operating system.

OS/2 Outline Fonts

The following Adobe Type 1 fonts are supplied with OS/2:

Family Name	Face Name
Times New Roman	Times New Roman Times New Roman Bold Times New Roman Bold Italic Times New Roman Italic
Helvetica	Helvetica Helvetica Bold Helvetica Bold Italic Helvetica Italic
Courier	Courier Courier Bold Courier Bold Italic Courier Italic
Symbol	Symbol

The Courier, Tms Rmn, and Swiss family fonts that were supplied with OS/2 release 1.1 and 1.2 are no longer supplied. Using one of the old names results in one of the new fonts listed above being used, as follows:

Old Family/Face Name	Font Used.
Roman/Tms Rmn	Times New Roman
Swiss/Helv	Helvetica

These fonts are provided in an efficient binary format for use by the OS/2 Adobe Type Manager. They are also provided in standard Type 1 format (PFB and AFM) for use with the OS/2 PostScript printer device driver.

Presentation Manager Bit Map Fonts

The following tables list all system bit map fonts available using the Graphics Programming Interface. The first table applies to hardware that does not conform to the International Standards Organization (ISO) 9241. (See [International Standards Organization \(ISO\) 9241](#) for more information on ISO 9241.) The second table lists the fonts supplied with OS/2 for IBM hardware that does conform to ISO 9241.

During system installation, the operating system determines the type of display adapter available on your computer and installs only the fonts which match the device resolution. Since additional device bit map fonts may be available on specific devices, you may have to install the correct bit map fonts if you change your display device after the operating system is installed.

Fonts Supplied for ISO 9241 Non-Conforming Hardware

The following information for each font is included in the table:

Points	This is the point size of the font, on a device whose resolution matches that of the font, (see "Device" below).	
Ave Wid	This is the average width in pels of alphabetic characters weighted according to US English letter frequencies.	
Max Wid	This is the maximum width in pels of all characters in the font. This field is not necessarily the maximum width of any character in the code page. It could be used to ensure that the horizontal space allocated on a display or printer is big enough to handle any character.	
Height	This is the height in pels of the font. This is the minimum number of rows of pels needed to output any character of the font on a given baseline. This field may be larger than necessary for a given code page. It could be used to ensure that the vertical space allocated on a display or printer is big enough to handle any character.	
Device	This is the X and Y resolution in pels per inch at which the font is intended to be used. Only those fonts which match the device resolution of the installed display driver are available on the system. If the installed display is changed, the install process will reinstall the proper font sets for the new adapter. The IBM devices whose device drivers report these resolutions are:	
	96 x 48	CGA
	96 x 72	EGA
	96 x 96	VGA and XGA (in 640 x 480 mode)
	120 x 120	8514/A and XGA (in 1024 x 768 mode)

Note: These values are approximate representations of the actual resolution, which in the case of displays depends on which monitor is attached. Consequently the point size of characters on the screen is also approximate.

The following table applies to hardware that does not conform to ISO 9241.

Family	Face Name	Points	Ave Wid	Max Wid	Height	Device
Courier	Courier	8	8	8	7	96x48
			8	8	10	96x72
			8	8	13	96x96
			9	9	16	120x120
		10	9	9	8	96x48
			9	9	12	96x72
			9	9	16	96x96
			12	12	20	120x120
		12	12	12	10	96x48
			12	12	15	96x72
			12	12	20	96x96
			15	15	25	120x120
System Pro- portional	System Proportional	8	6	20	8	96x48
		10	6	20	12	96x96
		10	6	20	16	96x96
		10	8	23	20	120x120
		11	10	23	23	120x120
System Mono- spaced	System Monospaced	8	8	8	8	96x48
		10	8	8	12	96x72
		10	8	8	16	96x96

Helv	Helv	10	9	9	20	120x120
		8	5	13	6	96x48
			5	13	10	96x72
			5	13	13	96x96
			6	14	16	120x120
		10	6	15	8	96x48
			6	14	12	96x72
			6	14	16	96x96
			7	20	20	120x120
		12	7	17	10	96x48
			7	17	15	96x72
			7	17	20	96x96
			9	21	25	120x120
		14	8	21	12	96x48
			8	21	18	96x72
			8	21	24	96x96
			11	26	29	120x120
		18	11	26	15	96x48
			10	26	22	96x72
			11	26	29	96x96
			13	34	36	120x120
		24	14	35	19	96x48
			14	35	28	96x72
			14	35	37	96x96
			18	45	46	120x120
Tms Rmn	Tms Rmn	8	4	12	6	96x48
			4	13	10	96x72
			4	12	13	96x96
			5	14	16	120x120
		10	6	15	8	96x48
			5	14	12	96x72
			5	14	16	96x96
			7	19	20	120x120
		12	7	18	10	96x48
			6	18	15	96x72
			6	16	19	96x96
			8	23	23	120x120
		14	7	21	11	96x48
			7	21	16	96x72
			7	20	21	96x96

	10	26	27	120x120
18	10	26	14	96x48
	10	26	20	96x72
	10	26	27	96x96
	12	34	33	120x120
24	14	35	18	96x48
	13	35	26	96x72
	13	35	35	96x96
	16	46	43	120x120

Fonts Supplied for ISO 9241 Conforming Hardware

The following table lists the fonts and sizes that have been tested and certified as passing the ISO 9241 black text on white background criteria for the three IBM displays that conform to the standard. These displays are:

- 9515 - A 14 inch XGA display.
- 9517 - A 17 inch XGA display.
- 9518 - A 14 inch VGA display.

See [International Standards Organization \(ISO\) 9241](#) for information on ISO 9241.

The following information about each font is also included in the table:

P	The point size of the font.									
AW	The average character width in pels in the font.									
MW	The maximum character width in pels in the font.									
HE	The height in pels of the font (maximum baseline extent).									
Device	The X and Y resolution in pels per inch on the device the font is intended to be used. The IBM devices whose device drivers report these resolutions are:									
	96 x 96						VGA and XGA (in 640 x 480 mode)			
	120 x 120						XGA (in 1024 x 768 mode)			

Family Name	Face Name	P	AW	MW	HE	Device	9515	9517	9518
Courier	Courier ISO	8	8	8	13	96 96	No	No	No
		8	10	10	16	120 120	No	No	n/a
		9	8	8	15	96 96	Yes	Yes	Yes
		10	10	10	16	96 96	Yes	Yes	Yes
		10	12	12	20	120 120	No	No	n/a
		12	12	12	20	96 96	Yes	Yes	Yes
Helv	Helv ISO	12	15	15	25	120 120	Yes	Yes	n/a
		8	5	13	13	96 96	No	No	No
		8	7	14	16	120 120	No	No	n/a
		9	6	13	15	96 96	Yes	Yes	Yes
		9	8	20	21	120 120	Yes	Yes	n/a
		10	7	14	16	96 96	Yes	Yes	Yes
		10	9	20	21	120 120	Yes	Yes	n/a
		12	9	17	20	96 96	Yes	Yes	Yes
		12	10	21	25	120 120	Yes	Yes	n/a
		14	10	21	24	96 96	Yes	Yes	Yes
		14	12	26	29	120 120	Yes	Yes	n/a
		18	12	26	29	96 96	Yes	Yes	Yes
		18	15	34	36	120 120	Yes	Yes	n/a
		24	14	34	36	96 96	Yes	Yes	Yes
		24	19	45	46	120 120	Yes	Yes	n/a
Tms Rmn	Tms Rmn ISO	8	5	12	13	96 96	No	No	No
		8	7	15	16	120 120	No	No	n/a
		9	6	12	15	96 96	Yes	Yes	Yes
		10	7	14	16	96 96	Yes	Yes	Yes
		10	8	17	19	120 120	No	Yes	n/a
		12	8	16	19	96 96	Yes	Yes	Yes

		12	10	23	22	120	120	Yes	Yes	n/a
		14	9	23	22	96	96	Yes	Yes	Yes
		14	11	26	27	120	120	Yes	Yes	n/a
		18	11	26	27	96	96	Yes	Yes	Yes
		18	14	34	34	120	120	Yes	Yes	n/a
		24	14	34	34	96	96	Yes	Yes	Yes
		24	17	46	43	120	120	Yes	Yes	n/a
System	System	9	6	13	15	96	96	Yes	Yes	Yes
Proportional	Proportional	10	6	20	16	96	96	Yes	Yes	Yes
		10	8	23	20	120	120	No	Yes	n/a
		12	10	23	22	120	120	Yes	Yes	n/a
System Mono-spaced	System Mono-spaced	10	8	8	16	96	96	Yes	Yes	Yes
		10	10	10	21	120	120	Yes	Yes	n/a

See [International Standards Organization \(ISO\) 9241](#) for more information on ISO 9241.

International Standards Organization (ISO) 9241

ISO 9241 is an international standard covering health and safety in the work place for users of visual display terminals. Part 3 of this standard covers clarity and legibility of text displayed on computer screens; it places requirements on minimum sizes and luminance contrast. The presence of the FM_SEL_ISO9241_TESTED flag in the [FONTMETRICS](#) structure indicates that the font has been tested for ISO compliance.

Note: While the fonts were primarily tested for meeting the ISO standard, they have also been designed to meet the German standard DIN 66 234. Where the two standards differ, the fonts have been designed to meet the stricter requirement.

The FM_ISO_xxx flags indicate the results of the test on the three IBM displays that conform to the standard. These are the IBM 9515, 9517, and 9518 color displays at the supported resolutions of 640 x 480 and 1024 x 768. To determine whether a non-IBM display complies with ISO 9241, contact the manufacturer. The current display type can be established using VioGetConfig.

In order for applications to meet the standard, they have to ensure that they use only fonts that have been tested and passed. You can determine this by examining the new FM_SEL_ISO9241_TESTED flag in the *fsSelection* parameter in the [FONTMETRICS](#) structure, the FM_ISO_xxx flags and the *sXDeviceRes* and *sYDeviceRes* fields in the structure.

See [Fonts Supplied with the OS/2 Operating System](#) for the table describing ISO 9241 compliant fonts.

Initialization File Information

Initialization files include information about printers, queues, and system preferences set by the user from the control panel. Applications can query this information by using the [PrfQueryProfileData](#), [PrfQueryProfileInt](#), [PrfQueryProfileSize](#), and [PrfQueryProfileString](#) functions.

All data in initialization files is accessed by a two-level hierarchy of application name, and key name within an application. Presentation Manager system data is keyed off "applications" that have names starting with PM_.

The application name/key name combinations that applications may need to use are listed below, together with the definition of the corresponding data.

Note: Information that is prefixed with PM_SPOOLERxxxx can not always be modified directly: The spooler validates all attempts to write information to the INI file that it depends on.

Application name	"PM_ControlPanel"
Key name	"Beep"
Type	integer
Content/value	1 or 0.
Application name	"PM_ControlPanel"
Key name	"LogoDisplayTime"
Type	integer

Content/value	-1 <= time <= 32767 milliseconds. Indefinite display No display Timed display	-1 0 >0
Application name	"PM_ControlPanel"	
Key name	"cxDoubleClick"	
Type	integer	
Content/value	SV_CXDBLCLK size in pels.	
Application name	"PM_ControlPanel"	
Key name	"cyDoubleClick"	
Type	integer	
Content/value	SV_CYDBLCLK size in pels.	
Application name	"PM_ControlPanel"	
Key name	"cxMotionStart"	
Type	integer	
Content/value	SV_CXMOTIONSTART size in pels.	
Application name	"PM_ControlPanel"	
Key name	"cyMotionStart"	
Type	integer	
Content/value	SV_CYMOTIONSTART size in pels.	
Application name	"PM_National"	
Key name	"iCountry"	
Type	integer	
Content/value	country code: Arabic Australian Belgian Canadian-French Danish Finnish French German Hebrew Italian Japanese Korean Latin-American Netherlands Norwegian Portuguese Simpl. Chinese Spanish Swedish Swiss Trad. Chinese UK-English US-English Other country	785 61 32 2 45 358 33 49 972 39 81 82 3 31 47 351 86 34 46 41 88 44 1 0.
Application name	"PM_National"	
Key name	"iDate"	
Type	integer	
Content/value	0=MDY; 1=DMY; 2=YMD.	
Application name	"PM_National"	
Key name	"iCurrency"	
Type	integer	
Content/value	Values have the following meanings: 0 1 2 3	Prefix, no separator Suffix, no separator Prefix, 1 character separator Suffix, 1 character separator.
Application name	"PM_National"	
Key name	"iDigits"	
Type	integer	
Content/value	n = number of decimal digits.	

Application name	"PM_National"
Key name	"iTime"
Type	integer
Content/value	0 = 12-hour clock; 1 = 24-hour clock.

Application name	"PM_National"
Key name	"iLzero"
Type	integer
Content/value	0 = no leading zero; 1 = leading zero.

Application name	"PM_National"
Key name	"s1159"
Type	string
Content/value	"am" for example. 3 chars max.

Application name	"PM_National"
Key name	"s2359"
Type	string
Content/value	"pm" for example. 3 chars max.

Application name	"PM_National"
Key name	"sCurrency"
Type	string
Content/value	"\$" for example. 3 chars max.

Application name	"PM_National"
Key name	"sThousand"
Type	string
Content/value	"," for example. 1 char max.

Application name	"PM_National"
Key name	"sDecimal"
Type	string
Content/value	"." for example. 1 char max.

Application name	"PM_National"
Key name	"sDate"
Type	string
Content/value	"/" for example. 1 char max.

Application name	"PM_National"
Key name	"sTime"
Type	string
Content/value	":" for example. 1 char max.

Application name	"PM_National"
Key name	"sList"
Type	string
Content/value	"," for example. 1 char max.

Application name	PM_Fonts
Key name	
Type	string
Content/value	fully-qualified drive:\path\filename.ext.

Application name	"PM_SPOOLER"
Key name	"QUEUE"
Type	string
Content/value	<Queue name>;

where: <Queue name> is the name of the default queue (might be NULL). This must be a key name for the PM_SPOOLER_QUEUE application.

Application name	"PM_SPOOLER"
Key name	" PRINTER"
Type	string
Content/value	<Printer name>;

where: <Printer name> is the name of the default printer (might be NULL).

Note: Use the [SplQueryDevice](#) and [SplQueryQueue](#) functions to retrieve the spooler configuration data.

Interchange File Format

A metafile is a file in which graphics are stored. The file is application-created, and it contains the graphics orders generated from those GPI calls that are valid in a metafile. Metafiled graphics can be reused by the application that created them. They can also be made available to other applications at the same, or at a different, workstation.

This section describes the restrictions which apply when generating the metafile and gives detail of the overall structure. For the graphics orders descriptions, see "Graphics Orders" in the *Graphics Programming Interface Programming Reference*.

Metafile Restrictions

The following restrictions apply to the generation of all metafiles, and also to the generation of a PM_Q_STD print file to a device:

- If GpiWCBitBlt or GpiBitBlt is used to copy a bit map to a device context in an application, the application should not delete that bit map handle with GpiDeleteBitmap before the device context is closed (metafile is closed).
- GpiSetPS must not be used.
- GpiSetPageViewport is ignored.

The following section lists some general rules that must be followed when creating a metafile that is to be acceptable to SAA-conforming implementations, or replayed into a presentation space that is in **draw-and-retain** or **retain** mode (see "GpiSetDrawingMode" in *Graphics Programming Interface Programming Reference*).

- These items must be established or defaulted before any drawing occurs to the graphics presentation space, and not changed subsequently:
 - The graphics field (GpiSetGraphicsField). For an SAA-conforming metafile, the graphics field must be defaulted or set to no clipping.
 - The code page for the default character set (GpiSetCp).
 - The color table or palette (GpiCreateLogColorTable or GpiCreatePalette). The size of the color table must not exceed 31KB (KB equals 1024 bytes).
 - The default viewing transform (GpiSetDefaultViewMatrix).
 - The setting of the draw controls (GpiSetDrawControl). DCTL_DISPLAY must be defaulted or set ON.
 - The default values of attributes (see "GpiSetDefAttrs" in the *Graphics Programming Interface Programming Reference*), viewing limits (see "GpiSetDefViewingLimits" in the *Graphics Programming Interface Programming Reference*), primitive tag (see "GpiSetDefTag" in the *Graphics Programming Interface Programming Reference*) and arc parameters (see "GpiSetDefArcParams" in the *Graphics Programming Interface Programming Reference*).
- These calls should not be used:
 - GpiBitBlt
 - GpiDeleteSetId (note that this means that local identifiers cannot be used again within the picture)
 - GpiErase
 - GpiExcludeClipRectangle
 - GpiIntersectClipRectangle
 - GpiOffsetClipRegion
 - GpiPaintRegion
 - GpiResetPS
 - GpiSetClipRegion
 - GpiSetPel
 - GpiSetPS
 - [DevEscape](#) (for an escape which is metafiled).
- GpiCreateLogFont must not redefine a local identifier that has previously been used within the picture.
- The metafile context must not be reassociated.
- If a bit map is used as the source of a GpiWCBitBlt operation, or as an area-fill pattern, it must not be modified or deleted (GpiDeleteBitmap) before the metafile is closed.
- Only these foreground mixes must be used (see "GpiSetMix" in the *Graphics Programming Interface Programming Reference*):

- FM_DEFAULT
 - FM_OR
 - FM_OVERPAINT
 - FM_LEAVEALONE
- Only these background mixes must be used (see "GpiSetBackMix" in the *Graphics Programming Interface Programming Reference*):
 - BM_DEFAULT
 - BM_OVERPAINT
 - BM_LEAVEALONE
 - If palettes are used (see "GpiCreatePalette" in the *Graphics Programming Interface Programming Reference*): the palette that is metafiled is the one in force when the metafile device context is dissociated from the (final) presentation space. If the palette is changed during the course of the picture (using GpiSetPaletteEntries), it must therefore only be with incremental additions.

Note: There is no restriction concerning the use of primitives outside segments. These are metafiled in segment(s) with zero identifier.

Metafile Data Format

This section describes the format of the data in a metafile, as it would be stored in an OS/2 disk file.

Metafile data is stored as a sequence of structured fields. Each structured field starts with an eight-byte header consisting of a two-byte *length* field and a three-byte *identifier* field. These are followed by a one-byte *flags* field and a two-byte *segment sequence number* field.

The length field contains a count of the total number of bytes in the structured field, including the length field. The identifier field uniquely identifies the type of the structured field.

The flags and segment sequence number fields are always zero.

Following the header are positional parameters that are optional and dependent on the particular structured field.

Following the positional parameters are non-positional parameters called *triplets*. These are self-defining parameters and consist of a one-byte *length* field, followed by a one-byte *identifier* field, followed by the data of the parameter.

The length field contains a count of the total number of bytes in the triplet, including the length and identifier fields. The identifier field identifies uniquely the type of the triplet.

A metafile is structured into a number of different functional components; for example, document and graphics object. Each component comprises a number of structured fields, and is delimited by "begin-component" and "end-component" structured fields. Structured fields marked as *required*, inside an *optional* structured field bracket, are required if the containing bracket is present.

The graphics orders that describe a picture occur in the *graphics data* structured field. See [Structured Field Formats](#) for more information.

Structured Field Formats

Following is the format of the various structured fields:

Begin Document

Structured Field Introducer (BDT): required

0-1	Length 0xn+1E
2-4	BDT 0xD3A8A8
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters

0-7	Document name C'0000 0001'
8	Architecture version 0x00
9	Document security 0x00

Triplets (all required)

0	Length 0x05
1	Triplet Id 0x18
2	Interchange set type 0x03 (resource document)
3-4	Base set definition 0x0C00 (level 12, version 0)
0	Length 0x06
1	Triplet Id 0x01
2-5	GCID
0	Length 0xn+1
1	Triplet Id 0x65
2-n	Comment, used for metafile description of up to 252 bytes.

Begin Resource Group (BRG): required

Structured Field Introducer

0-1	Length 0x0010
2-4	BRG 0xD3A8C6
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters

0-7	Resource group name C'0000 0002'
-----	----------------------------------

Begin Color Attribute (BCA) Table: required

Structured Field Introducer

0-1	Length 0x0010
2-4	BCA 0xD3A877
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters

0-7	Color table name C'0000 0004'
-----	-------------------------------

Color Attribute Table (CAT): required

Structured Field Introducer

0-1	Length 0xn+8
2-4	CAT 0xD3B077
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters

Base Part (required)

0	Flags		
	0	Reserved B'0'	
	1	Reset	
		B'0'	Do not reset to default
		B'1'	Do reset to default
	2-7	Reserved B'000000'	
1	Reserved 0x00		
2	LCTID 0x00		

Element list(s) (triple generating) are mutually-exclusive. One or other is required.

Element List (repeating)

0	Length of this parameter
1	Type 0x01: element list

2	Flags 0x00: reserved
3	Format 0x01 RGB
4-6	Starting Index (Top Byte Truncated)
7	Size of RGB component1 0x08
8	Size of RGB component2 0x08
9	Size of RGB component3 0x08
10	Number of bytes in each following color triple 0x04
11-m	Color triples

Triple Generating

0	Length of this parameter 0x0A
1	Type 0x02: bit generator
2	Flags 0 ABFlag B'0' Normal 1-7 Reserved B'0000000'
3	Format 0x01 RGB 4-6 Starting index (top byte truncated)
7	Size of RGB component1 0x08
8	Size of RGB component2 0x08
9	Size of RGB component3 0x08

End Color Attribute (ECA) Table: required

Structured Field Introducer

0-1	Length 0x0010
2-4	ECA 0xD3A977
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters

0-7	Color table name C'0000 0004'
-----	-------------------------------

Begin Image Object (BIM): optional, repeating

Structured Field Introducer

0-1	Length 0x0010
2-4	BIM 0xD3A8FB
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters

0-7	Image name C'xxxx xxxx'
-----	-------------------------

Begin Resource Group (BRG): optional

Structured Field Introducer

0-1	Length 0x0010
2-4	BRG 0xD3A8C6
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters

0-7	Resource group name C'xxxx xxxx'
-----	----------------------------------

Color Attribute Table (BCA): optional

Structured Field Introducer

0-1	Length 0x0010
2-4	BCA 0xD3A877
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters

0-7 Color table name C'xxxx xxxx'

Color Attribute Table (CAT): required

Structured Field Introducer

0-1 Length
2-4 CAT 0xD3B077
5 Flags 0x00
6-7 Segment sequence number 0x0000

Parameters

Base Part

0 Flags 0x00
1 Reserved 0x00
2 LUTID

Element List (repeating)

0 Length of this parameter
1 Type 0x01: element list
2 Flags 0x00: reserved
3 Format 0x01: RGB
4-6 Starting index
(top byte truncated)
7 Size of RGB component1 0x08
8 Size of RGB component2 0x08
9 Size of RGB component3 0x08
10 Number of bytes in each following color triple 0x03
11-n Color triples

End Color Attribute Table (ECA): required if BCA present

Structured Field Introducer

0-1 Length 0x0010
2-4 ECA 0xD3A977
5 Flags 0x00
6-7 Segment sequence number 0x0000

Parameters

0-7 Color Table name C'xxxx xxxx'

End Resource Group (ERG): required if BRG present

Structured Field Introducer

0-1 Length 0x0010
2-4 ERG 0xD3A9C6
5 Flags 0x00
6-7 Segment sequence number 0x0000

Parameters

0-7 Resource Group name C'xxxx xxxx'

Begin Object Environment Group (BOG): optional

Structured Field Introducer

0-1 Length 0x0010
2-4 BOG 0xD3A8C7
5 Flags 0x00
6-7 Segment sequence number 0x0000

Parameters

0-7 Object environment group name C'xxxx xxxx'

Map Color Attribute (MCA) Table: required**Structured Field Introducer**

0-1	Length 0x001A
2-4	MCA 0xD3AB77
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters

0-1	Length
-----	--------

Triplet (required)

0	Length 0x0C
1	Triplet type: fully qualified name 0x02
2	Type: ref to Begin Resource Object 0x84
3	ID 0x00
4-11	Color table name C'xxxx xxxx'

lcid (required)

0	Length 0x04
1	Triplet type: resource local ID 0x24
2	Type color table resource 0x07
3	Local identifier (LUT-ID) 0x01

End Object Environment Group (EOG): required if BOG present**Structured Field Introducer**

0-1	Length 0x0010
2-4	EOG 0xD3A9C7
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters

0-7	Object Environment Group name C'xxxx xxxx'
-----	--

Image Data Descriptor (IDD): required**Structured Field Introducer**

0-1	Length 0x0011
2-4	IDD 0xD3A6FB
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters

0	Unit of measure:
	0x00 tens of inches
	0x01 tens of centimeters
1-2	X resolution image points / UOM
3-4	Y resolution image points / UOM
5-6	X extent of image PS
7-8	Y extent of image PS

Image Picture Data (IPD): required**Structured Field Introducer**

0-1	Length
2-4	IPD 0xD3EEFB
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters (all required and in this order, except that only one of Image LUT-ID and IDE structure is present)

Begin Segment

0	Type 0x70: begin segment
1	Length of following 0x00

Begin Image Content

0	Type 0x91: Begin Image Content
1	Length of following 0x01
2	Format 0xFF

Image Size

0	Type 0x94: image size
1	Length of following 0x09
2	Units of measure 0x02: logical
3-4	Horizontal resolution
5-6	Vertical resolution
7-8	Height in pels
9-10	Width in pels

Image Encoding

0	Type 0x95: image encoding
1	Length of following 0x02
2	Compression algorithm 0x03: none
3	Recording algorithm 0x03: bottom-to-top

Image IDE-Size

0	Type 0x96: image IDE-Size
1	Length of following 0x01
2	Number of bits per element

Image LUT-ID (For bit maps with other than 24 bits per pel)

0	Type 0x97 Image LUT-ID
1	Length of following 0x01
2	LUT-ID

IDE Structure (For bit maps with 24 bits per pel)

0	Type 0x9B: IDE structure	
1	Length of following 0x08	
2	Flags:	
0	ABFlag	Normal (Additive)
	B'0'	
1-7	Reserved B'0000000'	
3	Format	
0x01	RGB	
4-6	Reserved 0x000000	
7	Size of element 1	
8	Size of element 2	
9	Size of element 3	

Image Picture Data (IPD): required, repeating

Structured Field Introducer

0-1	Length
2-4	IPD 0xD3EEFB
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters

Image Data

0-1	Type 0xFE92: image data
2-3	Length of following
4-n	Image data (scan lines of bit maps)

End Image Content (required, only present in last Image Picture Data)

0	Type 0x93: End Image Content
1	Length of following 0x00

End Segment (required, only present in last Image Picture Data)

0	Type 0x71: end segment
1	Length of following 0x00

End Image Object (EIM): required if BIM present

Structured Field Introducer

0-1	Length 0x0010
2-4	EIM 0xD3A9FB
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters

0-7	Image name C'xxxx xxxx'
-----	-------------------------

Begin Graphics Object (BGR): required

Structured Field Introducer

0-1	Length 0x0010
2-4	BGR 0xD3A8BB
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters

0-7	Graphics object name C'0000 0007'
-----	-----------------------------------

Begin Object Environment Group (BOG): optional

Structured Field Introducer

0-1	Length 0x0010
2-4	LOG 0xD3A8C7
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters

0-7	Object Environment Group name C'0000 0007'
-----	--

Map Color Attribute Table (MCA): required

Structured Field Introducer

0-1	Length 0x0016
2-4	MCA 0xD3AB77
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters

0-1	Length
-----	--------

Triplet (required)

0	Length 0x0C
1	Triplet type: fully qualified name 0x02
2	Type: ref to Begin Resource Object 0x84
3	ID 0x00
4-11	Color table name C'0000 0004'

Map Coded Font (MCF): required, for default font

Structured Field Introducer

0-1	Length 0x20
-----	-------------

2-4	MCF 0xD3AB8A
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters

0-1	Length
-----	--------

Triplets (required)

Font name

0	Length 0x0C
1	Triplet type: fully qualified name 0x02
2	Type: ref to coded font 0x84
3	ID 0x00
4-11	Coded font name: C'nnxx xxxx' where n is 0xFF

lcid

0	Length 0x04
1	Triplet type: Resource Local ID 0x24
2	Type: Coded Font Resource 0x05
3	Local identifier (LCID) 0x00

Font Binary GCID

0	Length 0x06
1	Triplet type: Font Binary GCID 0x20
2-5	GCID

Map Coded Font (MCF): optional, repeating, for loaded fonts

Structured Field Introducer

0-1	Length 0x58
2-4	MCF 0xD3AB8A
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters

0-1	Length
-----	--------

Triplets (required)

Font name

0	Length 0x0C
1	Triplet type: fully qualified name 0x02
2	Type: ref to coded font 0x84
3	ID 0x00
4-11	Coded font name

lcid

0	Length 0x04
1	Triplet type: Resource Local ID 0x24
2	Type: coded font resource 0x05
3	Local identifier (LCID)

Font Attributes

0	Length 0x14
1	Triplet type: Font Descriptor 0x1F
2	Weight Class
3	Width Class
4-5	Font Height
6-7	Char Width
8	Descript Flags
9	Usage Codes
10	Family

11	Activity Class
12	Font Quality
13-14	CAP Height
15-16	X Height
17-18	Line Density
19	Use Flags

Font Binary GCID

0	Length 0x06
1	Triplet type: Font Binary GCID 0x20
2-5	GCID

Font Typeface

0	Length 0x24
1	Triplet type: fully qualified name 0x02
2	Type: ref to font typeface 0x08
3	ID 0x00
4-35	Font typeface C'xxx..xxx'

Map Data Resource (MDR): optional, repeating

Structured Field Introducer

0-1	Length 0x1D
2-4	MDR 0xD3ABC3
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters

0-1	Length
-----	--------

Triplets (required)

Bit-map Name

0	Length 0x0C
1	Triplet type: fully qualified name 0x02
2	Type: ref to Image Object 0x84
3	ID 0x00
4-11	Image name C'xxxx xxxx'

Extended Resource Icid

0	Length 0x07
1	Triplet type: Extended Resource Local ID 0x22
2	Type: Image Resource 0x10
3-6	Bit-map handle

End Object Environment Group (EOG): required if BOG present

Structured Field Introducer

0-1	Length 0x0010
2-4	EOG 0xD3A9C7
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters

0-7	Object Environment Group name C'0000 0007'
-----	--

Graphics Data Descriptor (GDD): required

Structured Field Introducer

0-1	Length 0xn timer
2-4	GDD 0xD3A6BB
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters (all required and in this order)

0	0xF7 Specify GVM Subset		
1	Length of following data 0x07		
2	0xB0 drawing order subset		
3-4	0x0000		
5	0x23 Level 3.2		
6	0x01 Version 1		
7	Length of following field 0x01		
8	Coordinate types in data		
	0x04	Intel16	
	0x05	Intel32	
0	0xF6 Set Picture Descriptor		
1	Length of following data		
2	Flags		
	0	B'0' Picture in 2D	
	1	Picture Dimensions	
		B'0'	Not absolute (PU_ARBITRARY PS)
		B'1'	Absolute (example: PU_TWIPS PS)
	2	Picture Elements	
		B'0'	Not pels
		B'1'	Pels (PU_PELS PS)
			(Bit 1 must also be set)
	3-7	B'00000'	
3	0x00 Reserved		
4	Picture frame size coordinate type		
	0x04	Intel16	
	0x05	Intel32	
5	UnitsOfMeasure		
	0x00	Ten inches	
	0x01	Decimeter	
6-11 or 6-17	(2 or 4 bytes) Resolution.		
		GPS Units / UOM on x axis	
		GPS Units / UOM on y axis	
		GPS Units / UOM on z axis	
12-23 or 18-41	(2 or 4 bytes) Window Size.		
		GPS X left, X right	
		GPS Y bottom, Y top	
		GPS Z near, Z far	
0	0x21 Set Current Defaults		
1	Length of following data		
2	Set Default Parameter Format 0x08		
3-4	Mask 0xE000		
5	Names 0x8F		
6	Coordinates		
	0x00	Picture in 2D	
7	Transforms		
	0x04	Intel16	
	0x05	Intel32	
8	Geometrics		
	0x04	Intel16	
	0x05	Intel32	
0	0x21 Set Current Defaults		
1	Length of following data		
2	Set default viewing transform 0x07		
3-4	Mask 0xCC0C		
5	Names 0x8F		
6-n	M11, M12, M21, M22, M41, M42	Matrix elements	
0	0x21 Set Current Defaults		
1	Length of following data		
2	Set default line attributes 0x01		
3-4	Mask - OR of as many of the following bits as are required:		
	0x8000	Line type	
	0x4000	Line width	
	0x2000	Line end	
	0x1000	Line join	
	0x0800	Stroke width	
	0x0008	Line color	
	0x0002	Line mix	

5	Flags	
	0x0F	Set indicated default attributes to initial values. (Data field is not present in this instance).
	0x8F	Set indicated default attributes to specified values.
6-n	Data - data values as required, in the following order if present.	
	No space is reserved for attributes for which the corresponding mask flag was not set.	
	(1 byte)	- Line type
	(1 byte)	- Line width
	(1 byte)	- Line end
	(1 byte)	- Line join
	(G bytes)	- Stroke width
	(4 bytes)	- Line color
	(1 byte)	- Line mix (G=2 or 4 depending on the Geometrics parameter of Set Default Parameter Format)
0	0x21 Set Current Defaults	
1	Length of following data	
2	Set Default Character Attributes 0x02	
3-4	Mask - OR of as many of the following bits as are required:	
	0x8000	Character angle
	0x4000	Character box
	0x2000	Character direction
	0x1000	Character precision
	0x0800	Character set
	0x0400	Character shear
	0x0040	Character break extra
	0x0020	Character extra
	0x0008	Character color
	0x0004	Character background color
	0x0002	Character mix
	0x0001	Character background mix
5	Flags	
	0x0F	Set indicated default attributes to initial values. (Data field is not present in this case).
	0x8F	Set indicated default attributes to specified values.
6-n	Data - data values as required, in the following order if present.	
	No space is reserved for attributes for which the corresponding Mask flag was not set.	
	(2*G bytes)	- Character angle
	(2*G + 4 bytes)	- Character box
	(1 byte)	- Character direction
	(1 byte)	- Character precision
	(1 byte)	- Character set
	(2*G bytes)	- Character shear
	(4 bytes)	- Character break extra
	(4 bytes)	- Character extra
	(4 bytes)	- Character color
	(4 bytes)	- Character background color
	(1 byte)	- Character mix
	(1 byte)	- Character background mix (G=2 or 4 depending on the Geometrics parameter of Set Default Parameter Format)
0	0x21 Set Current Defaults	
1	Length of following data	
2	Set Default Marker Attributes 0x03	
3-4	Mask - OR of as many of the following bits as are required:	
	0x4000	Marker box
	0x1000	Marker precision
	0x0800	Marker set
	0x0100	Marker symbol
	0x0008	Marker color
	0x0004	Marker background color
	0x0002	Marker mix
	0x0001	Marker background mix
5	Flags	
	0x0F	Set indicated default attributes to initial values. (Data field is not present in this instance)
	0x8F	Set indicated default attributes to specified values.
6-n	Data - data values as required, in this order if present.	
	No space is reserved for attributes for which the corresponding Mask flag was not set.	
	(2*G bytes)	- Marker box
	(1 byte)	- Marker precision
	(1 byte)	- Marker set
	(1 byte)	- Marker symbol
	(4 bytes)	- Marker color
	(4 bytes)	- Marker background color

	(1 byte)	- Marker mix
	(1 byte)	- Marker background mix (G=2 or 4 depending on the Geometrics parameter of Set Default Parameter Format)
0	0x21 Set Current Defaults	
1	Length of following data	
2	Set Default Pattern Attributes 0x04	
3-4	Mask - OR of as many of the following bits as are required:	
	0x0800	Pattern set
	0x0100	Pattern symbol
	0x0080	Pattern reference point
	0x0008	Pattern color
	0x0004	Pattern background color
	0x0002	Pattern mix
	0x0001	Pattern background mix
	5	Flags
	0x0F	Set indicated default attributes to initial values. (Data field is not present in this instance)
	0x8F	Set indicated default attributes to specified values.
6-n	Data - data values as required, in this order if present. No space is reserved for attributes for which the corresponding Mask flag was not set.	
	(1 byte)	- Pattern set
	(1 byte)	- Pattern symbol
	(2*G bytes)	- Pattern reference point
	(4 bytes)	- Pattern color
	(4 bytes)	- Pattern background color
	(1 byte)	- Pattern mix
	(1 byte)	- Pattern background mix (G=2 or 4 depending on the Geometrics parameter of Set Default Parameter Format)
0	0x21 Set Current Defaults	
1	Length of following data	
2	Set Default Image Attributes 0x06	
3-4	Mask - OR of as many of these bits as are required:	
	0x0008	Image color
	0x0004	Image background color
	0x0002	Image mix
	0x0001	Image background mix
	5	Flags
	0x0F	Set indicated default attributes to initial values. (Data field is not present in this instance)
	0x8F	Set indicated default attributes to specified values.
6-n	Data - data values as required, in this order if present. No space is reserved for attributes for which the corresponding Mask flag was not set.	
	(4 bytes)	- Image color
	(4 bytes)	- Image background color
	(1 byte)	- Image mix
	(1 byte)	- Image background mix
0	0x21 Set Current Defaults	
1	Length of following data	
2	Set Default Viewing Window 0x05	
3-4	Mask - OR of as many of the following bits as are required:	
	0x8000	x left limit
	0x4000	x right limit
	0x2000	y bottom limit
	0x1000	y top limit
	5	Flags
	0x0F	Set indicated default attributes to initial values. (Data field is not present in this case).
	0x8F	Set indicated default attributes to specified values.
6-n	Data - data values as required, in the following order if present. No space is reserved for attributes for which the corresponding Mask flag was not set.	
	(2*G bytes)	- x left limit
	(2*G bytes)	- x right limit
	(2*G bytes)	- y bottom limit
	(2*G bytes)	- y top limit (G=2 or 4 depending on the Geometrics parameter of Set Default Parameter Format)
0	0x21 Set Current Defaults	
1	Length of following data	
2	Set Default Arc Parameters 0x0B	
3-4	Mask - OR of as many of the following bits as are required:	

	0x8000	P value
	0x4000	Q value
	0x2000	R value
	0x1000	S value
5	Flags	
	0x0F	Set indicated default attributes to initial values. (Data field is not present in this case).
	0x8F	Set indicated default attributes to specified values.
6-n	Data	- data values as required, in the following order if present. No space is reserved for attributes for which the corresponding Mask flag was not set.
	(G bytes)	- P value
	(G bytes)	- Q value
	(G bytes)	- R value
	(G bytes)	- S value (G=2 or 4 depending on the Geometrics parameter of Set Default Parameter Format)
0	0x21	Set Current Defaults
1		Length of following data
2		Set Default Pick Identifier 0x0C
3-4	Mask	- OR of as many of the following bits as are required:
	0x8000	Pick identifier
5	Flags	
	0x0F	Set indicated default attributes to initial values. (Data field is not present in this case).
	0x8F	Set indicated default attributes to specified values.
6-n	Data	- data values as required, in the following order if present. No space is reserved for attributes for which the corresponding Mask flag was not set.
	(4 bytes)	- Pick identifier
0	0xE7	Set Bit-map Identifier
1		Length of following data 0x07
2-3		Usage Flags 0x8000
4-7		Bit-map handle
8		Lcid

Graphics Data (GAD): optional, repeating

Structured Field Introducer

0-1	Length 0xn+9
2-4	GAD 0xD3EEBB
5	Flags 0x00
6-7	Segment sequence number 0x0000

Parameters (maximum length in one structured field is 32759)

Graphics Segment (optional, repeating)

Segment data (including the Begin Segment parameter) can be split at any point between successive Graphics Data structured fields.

0	0x70	Begin Segment
1		Length of following data 0x0E
2-5		Segment identifier
6		Segment attributes (1)
	0 B'1'	Invisible
	1 B'1'	Propagate invisibility
	2 B'1'	Detectable
	3 B'1'	Propagate detectability
	6 B'1'	Dynamic
	7 B'1'	Fast chaining
7		Segment attributes (2)
	0 B'1'	Non-chained
	3 B'1'	Prolog
8-9		Segment data length (low-order 2 bytes)
10-13		Reserved
14-15		Segment data length (high-order 2 bytes)
16-n		Graphics orders (see the <i>Graphics Programming Interface Programming Reference</i>)

End Graphics Object (EGR)

Structured Field Introducer

0-1	Length 0x0010
2-4	EGR 0xD3A9BB

5 Flags 0x00
6-7 Segment sequence number 0x0000

Parameters

0-7 Graphics object name C'0000 0007'

End Resource Group (ERG): required

Structured Field Introducer

0-1 Length 0x0010
2-4 ERG 0xD3A9C6
5 Flags 0x00
6-7 Segment sequence number 0x0000

Parameters

0-7 Resource Group name C'0000 0002'

End Document (EDT): required

Structured Field Introducer

0-1 Length 0x0010
2-4 EDT 0xD3A9A8
5 Flags 0x00
6-7 Segment sequence number 0x0000

Parameters

0-7 Document name C'0000 0001'

Resource Files

This section describes the syntax for the resource language using railroad syntax, and describes the formats used.

Resource files are used to build dialog templates, menu templates, accelerator tables, extended attribute association tables, keyboard scancode mapping tables, keyboard names and fonts. The files must be compiled before they can be used by application programs.

How to Read the Syntax Definitions

Throughout this reference, syntax is described using the following structure.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

The symbol indicates the beginning of a statement.

The symbol indicates that the statement syntax is continued on the next line.

The symbol indicates that a statement is continued from the previous line.

The ` symbol indicates the end of a statement.

Diagrams of syntactical units other than complete statements start with the symbol and end with the symbol.

- Required items appear on the horizontal line (the main path).

STATEMENT required_item `

- Optional items appear below the main path.

STATEMENT optional_item `

- If a choice can be made from two or more items, they appear vertically, in a stack.

If one of the items *must* be chosen, one item of the stack appears on the main path.

```
STATEMENT      required_choice1      `
                required_choice2
```

If choosing one of the items is optional, the entire stack appears below the main path.

```
STATEMENT      optional_choice1      `
                optional_choice2
```

- An arrow returning to the left above the main path indicates an item that can be repeated.

```
STATEMENT      repeatable_item      `
```

A repeat arrow above a stack indicates that a choice can be made from the stacked items, or a single choice can be repeated.

- Keywords appear in uppercase, for example:

```
PARM1
```

They must be spelled exactly as shown. Variables appear in all lowercase letters, for example:

```
parmx
```

They represent user-supplied names or values.

- If punctuation marks, parentheses, arithmetic operators, or such symbols are shown, they must be entered as part of the syntax.

Definitions Used in all Resources

The definitions used in all resources are defined in [Specification of Values](#) and [Resource Load and Memory Options](#).

Specification of Values

These rules apply to values specified in resources:

- Coordinates must be integers. There must be no space between the sign of the value and the value itself. For example, "-1" is allowed but "- 1" is not.
- Resource identifiers may be any of the following:
 - Positive integers
 - Names that resolve to positive integers
 - Strings (for some resources)
- Real values, containing a decimal point, cannot be used.

Resource Load and Memory Options

The following options define when each resource is loaded and how memory is allocated for each resource.

LOADOPTION	Resource loading options.	
	PRELOAD	Resource is loaded immediately.
	LOADONCALL	Resource is loaded when called.
MEMOPTION	Resource memory options.	
	FIXED	Resource remains at a fixed memory location.
	MOVEABLE	Resource can be moved if necessary to compact.
	DISCARDABLE	Resource can be discarded if no longer needed.
	SEGALIGN	Resources are aligned on 64K byte boundaries.

Resource Script File Specification

The resource script file defines the names and attributes of the resources to be added to the executable file of the application. The file consists of one or more resource statements that define the resource type and original file, if any. See the following for a description of the resource statements:

- [Single-Line Statements](#)
- [User-Defined Resources](#)
- [Directives](#)
- [Multiple-Line Statements.](#)

Single-Line Statements

The general form for all single-line statements is:

```
resourcetype  nameid          loadoption
               filename  `
memoption
```

resourcetype (**USHORT**)
One of the following keywords, specifying the type of resource to be loaded:

Keyword	Resource type
BITMAP	A bit-map resource is a custom bit map that an application intends to use in its screen display or as an item in a menu.
DEFAULTICON	This keyword installs the filename.ico icon definition under the ICON EA of the program file.

Example:

DEFAULTICON <filename.ico>

DLGINCLUDE	This statement tells the dialog editor which file to use as an include file for the dialogs in the resource file. The nameid is not applicable.
FONT	A font resource is a file containing a font.
ICON	An icon resource is a bit map defining the shape of the icon to be used for a given application.
POINTER	A pointer resource is a bit map defining the shape of the pointing device pointer on the display screen.

nameid (USHORT)

is either a unique name or an integer number identifying the resource. For a FONT resource, the **nameid** must be a number; it cannot be a name.

loadoption (LOADOPTION)

The default is LOADONCALL.

See [Resource Load and Memory Options](#) for a description of *LOADOPTION*.

memoption (MEMOPTION)

The default is MOVEABLE and DISCARDABLE for POINTER, ICON, and FONT resources. The default for BITMAP resources is MOVEABLE. The FIXED option overrides both MOVEABLE and DISCARDABLE. The SEGALIGN option can be specified independently of other options, if it is not present the default (for all resources) is that the resource is not aligned on a 64KB boundary.

See [Resource Load and Memory Options](#) for a description of *MEMOPTION*.

filename (PCH)

An ASCII string specifying the OS/2* name of the file containing the resource. A full path must be given if the file is not in the current working directory.

Example

```
POINTER "pointer" point.cur
POINTER "discardable pointer" DISCARDABLE point.cur
POINTER 10 custom.cur

ICON "desk" desk.ico
ICON "discardable desk" DISCARDABLE desk.ico
ICON 11 custom.ico

BITMAP "disk" disk.bmp
BITMAP "discardable disk" DISCARDABLE disk.bmp
BITMAP 12 custom.bmp

FONT 5 CMROMAN.FNT
```

User-Defined Resources

An application can also define its own resource. The resource can be any data that the application intends to use. A user-defined resource statement has the form:

```
resource-type    typeId    nameID

                loadoption    memoption    filename    `
```

typeID

Either a unique name or an integer number identifying the resource type. If a number is given, it must be greater than 255. The type

numbers 1 through 255 are reserved for existing and future predefined resource types. Value 1000 is reserved for custom fonts.

nameID

Either a unique name or an integer number identifying the resource.

loadoption (**LOADOPTION**)

The default is LOADONCALL.

See [Resource Load and Memory Options](#) for a description of *LOADOPTION*.

memoption (**MEMOPTION**)

The default is MOVEABLE.

See [Resource Load and Memory Options](#) for a description of *MEMOPTION*.

filename

Can be either:

- An ASCII string specifying the OS/2* name of the file containing the cursor bit map. A full path must be given if the file is not in the current working directory.
- A BEGIN END data definition construct as follows:

```
BEGIN
data-definition [, data-definition] ...
.
.
.
END
```

For example, the RESOURCE statement might be written as follows:

```
RESOURCE MYRES "array"
BEGIN
    13L, 26L
END
```

When the resource compiler (RC.EXE) encounters one or more font resources, or any custom resource having type-id of 1000, it creates a font directory resource which it adds to the output binary data.

Example

```
RESOURCE MYRES    "array" DATA.RES
RESOURCE 300      14     CUSTOM.RES
```

RCDATA statement

The RCDATA statement is provided to allow an application to define a simple data resource.

```
RCDATA    id        loadoption    memoption

          '
          newline
BEGIN      data      END          `
```

id

Either a unique name or an integer number identifying the resource.

loadoption (**LOADOPTION**)

The default is LOADONCALL.

See [Resource Load and Memory Options](#) for a description of *LOADOPTION*.

memoption (**MEMOPTION**)

The default is MOVEABLE.

See [Resource Load and Memory Options](#) for a description of *MEMOPTION*.

data

A number or string.

Example:

```
RCDATA 4
BEGIN
    "Sample string."
    "TEST DATA."
    "A message."
END
```

Directives

The resource directives are special statements that define actions to perform on the file before it is compiled. The directives can assign values to names, include the contents of files, and control compilation of the file.

#include filename

rcinclude filename

These directives copy the contents of the file specified by **filename** into the resource before it is compiled. If **rcinclude** is used, the entire file is copied. If **#include** is used, only **#define** statements are copied.

Note: If an **rcinclude** is to be commented out, the open comment (*/**) must appear on the same line as the directive. **Filename** is an ASCII string. A full path must be given if the file is not in the current directory or in the directory specified by the INCLUDE environment variable. The file extensions .I and .TMP must not be used as these are reserved for system use.

The **filename** parameter is handled as a C string, and two backslashes must be given wherever one is expected in the path name (for example, root\sub.) or, a single forward slash (/) can be used instead of double backslashes (for example, root/sub.)

Example:

```
#include "wincalls.h"

MENU PenSelect
BEGIN
    MENUITEM "black pen", BLACK_PEN
END
```

Files included in resource script files constants that use **#define** statements may not include any casting of those constants that are used in the resource script. The resource compiler does not parse this casting syntax. For example, the following statement may not be included:

```
#define IDBUTTON1 (USHORT) 3
```

If casting is required for C source compilation, you may use two statements such as:

```
#define IDBUTTON1 3
```

```
#define CSRC_IDBUTTON1    ((USHORT)IDBUTTON1)
```

#define name value

This directive assigns the given value to **name**. All subsequent occurrences of **name** are replaced by the value.

name is any combination of letters, digits, or punctuation.

value is any integer, character string, or line of text.

Example:

```
#define    nonzero        1
#define    USERCLASS      "MyControlClass"
```

#undef name

This directive removes the current definition of **name**. All subsequent occurrences of **name** are processed without replacement.

name is any combination of letters, digits, or punctuation.

Example:

```
#undef    nonzero
#undef    USERCLASS
```

#ifdef name

This directive performs a conditional compilation of the resource file by checking the specified name. If the name has been defined using a #define directive, #ifdef directs the resource compiler to continue with the statement immediately after it. If the name has not been defined, #ifdef directs the compiler to skip all statements up to the next #endif directive.

name is the name to be checked by the directive.

Example:

```
#ifdef Debug
FONT 4 errfont.fnt
#endif
```

#ifndef name

This directive performs a conditional compilation of the resource file by checking the specified name. If the name has not been defined or if its definition has been removed using the #undef directive, #ifndef directs the resource compiler to continue processing statements up to the next #endif, #else, or #elif directive, then skip to the statement after the #endif. If the name is defined, #ifndef directs the compiler to skip to the next #endif, #else, or #elif directive.

name is the name to be checked by the directive.

Example:

```
#ifndef Optimize
FONT 4 errfont.fnt
#endif
```

#if constant expression

This directive performs a conditional compilation of the resource file by checking the specified constant-expression. If the constant-expression is nonzero, `#if` directs the resource compiler to continue processing statements up to the next `#endif`, `#else`, or `#elif` directive, then skip to the statement after the `#endif`. If the constant-expression is zero, `#if` directs the compiler to skip to the next `#endif`, `#else`, or `#elif` directive.

constant expression is a defined name, an integer constant, or an expression consisting of names, integers, and arithmetic and relational operators.

Example:

```
#if Version<3
FONT 4 errfont.fnt
#endif
```

#elif constant expression

This directive marks an optional clause of a conditional compilation block defined by an `#ifdef`, `#ifndef`, or `#if` directive. The directive carries out conditional compilation of the resource file by checking the specified constant-expression. If the constant-expression is nonzero, `#elif` directs the resource compiler to continue processing statements up to the next `#endif`, `#else`, or `#elif` directive, then skip to the statement after the `#endif`. If the constant-expression is zero, `#elif` directs the compiler to skip to the next `#endif`, `#else`, or `#elif` directive. Any number of `#elif` directives can be used in a conditional block.

constant expression Is a defined name, an integer constant, or an expression consisting of names, integers, and arithmetic and relational operators.

Example:

```
#if Version<3
FONT 4 italic.fnt
#elif Version<7
FONT 4 bold.fnt
#endif
```

#else

This directive marks an optional clause of a conditional compilation block defined by an `#ifdef`, `#ifndef`, or `#if` directive. The `#else` directive must be the last directive before `#endif`.

Example:

```
#ifdef Debug
FONT 4 italic.fnt
#else
FONT 4 bold.fnt
#endif
```

#endif

This directive marks the end of a conditional compilation block defined by an `#ifdef`, `#ifndef`, or `#if` directive. One `#endif` is required for each `#ifdef`, `#ifndef`, and `#if` directive.

Multiple-Line Statements

This sections covers [Code Page Flagging](#), [Keyboard Resources](#), and the following multiple-line statements:

- [ACCELTABLE Statement](#)
- [ASSOCTABLE Statement](#)

- [MENU Statement](#)
- [STRINGTABLE Statement](#)
- [Dialog and Window Template Statements](#)

Code Page Flagging

The CODEPAGE statement may be placed within the source, to set the code page used for these resources:

- ACCELTABLE
- MENU
- STRINGTABLE
- DIALOGTEMPLATE and WINDOWTEMPLATE.

The CODEPAGE statement cannot be encoded within any other statement. All items following a CODEPAGE statement are assumed to be in that code page. The code page is encoded in the resource, and the data in the resource is assumed to be in the specified code page. However, no checking is performed.

These code pages can be specified:

- 437
- 850
- 860
- 863
- 865.

If the code page is not specified, code page 850 is assumed.

Keyboard Resources

RT_FKALONG (=17), is defined in BSEDOS.H, and the resource compiler (RC.EXE) recognizes **FKALONG**. This type identifies a 256-byte table, that can be used for either primary or secondary scan-code mapping.

The resource ID contains three bytes, the least significant byte identifying the type of scan-code mapping table as follows:

- | | |
|---|------------------------------|
| 0 | Primary scan-code mapping |
| 1 | Secondary scan-code mapping. |

The other two bytes are 0 for the primary mapping table, and the keyboard ID (as defined in PMWINP.H) for secondary mapping tables. This is to enable simple support to be provided for future keyboards with conflicting scan codes.

The primary scan-code mapping table in the interrupt handler is stored as a resource of this type. The secondary scan-code mapping table in the interrupt handler is also stored as a resource of this type.

Depending on which keyboard is attached, the resources are loaded when the system is initialized, and transferred to RING-0 byte arrays, where they can be accessed by the interrupt handler as necessary. A default primary scan-code mapping table is transferred if the resource cannot be loaded.

ACCELTABLE Statement

The ACCELTABLE statement defines a table of *accelerator* keys for an application.

An accelerator is a keystroke defined by the application to give the user a quick way to perform a task. The [WinGetMsg](#) function automatically translates accelerator messages from the application queue into [WM_COMMAND](#), [WM_HELP](#), or [WM_SYSCOMMAND](#) messages.

The ACCELTABLE statement has the form:


```

ACCELTABLE
    id          memoption

BEGIN

    ,

keyval  ,  cmd  ,  acceloption

END

```

id (USHORT)

The resource identifier. This is either an integer in the range of 1 through 65535 or a unique string enclosed in double quote marks.

memoption

Optional. It consists of the following keyword or keywords, specifying whether the resource is fixed or movable, and whether it can be discarded:

FIXED	Resource remains at a fixed memory location.
MOVEABLE	Resource can be moved if necessary to compact memory.
DISCARDABLE	Resource can be discarded if no longer needed.

See [Resource Load and Memory Options](#) for a description of *LOADOPTION*.

keyval (USHORT)

The accelerator character code. This can be either a constant or a quoted character. If it is a quoted character, the CHAR acceloption is assumed. If the quoted character is preceded with a caret character (^), a control character is specified as if the CONTROL **acceloption** had been used.

cmd (USHORT)

The value of the [WM_COMMAND](#), [WM_HELP](#), or [WM_SYSCOMMAND](#) message generated from the accelerator for the indicated key.

acceloption (BIT_16)

Defines the kind of accelerator.

The following options are available:

```

ALT
CHAR
CONTROL
HELP
LONEKEY
SCANCODE
SHIFT
SYSCOMMAND
VIRTUALKEY.

```

The VIRTUALKEY, SCANCODE, LONEKEY, and CHAR acceloptions specify the type of message that matches the accelerator. Only one of these options can be specified for each accelerator. For information on the corresponding KC_* values, see [WM_CHAR](#).

The **acceloptions** SHIFT, CONTROL, and ALT, cause a match of the accelerator only if the corresponding key is down.

If there are two accelerators that use the same key with different SHIFT, CONTROL, or ALT options, the more restrictive accelerator should be specified first in the table. For example, Shift-Enter should be placed before Enter.

The SYSCOMMAND **acceloption** causes the keystroke to be passed to the application as a [WM_SYSCOMMAND](#) message. The HELP **acceloption** causes the keystroke to be passed to the application as a [WM_HELP](#) message. If neither is specified, a [WM_COMMAND](#) message is used.

Example:

```

ACCELTABLE "MainAcc"
BEGIN
    VK_F1,101,HELP
    VK_F3,102,SYSCOMMAND
END

```

This generates a WM_HELP with value 101 from VIRTUALKEY accelerator F1 and a WM_SYSCOMMAND with value 102 from

VIRTUALKEY accelerator F3.

ASSOCTABLE Statement

The ASSOCTABLE statement defines the extended attributes (EA) for an application.

The ASSOCTABLE statement has the form:

```
ASSOCTABLE  assoctableid

BEGIN

    assocname,extensions      flags      ,      icon

END
```

The source for the ASSOCTABLE description is contained in the resource file for a particular project:

```
ASSOCTABLE  assoctableid
BEGIN
"association name", "extension", flags, icon filename
"association name", "extension", flags, icon filename
...
END
```

association name

Program recognizes data files of this EA TYPE. This is the same name found in the TYPE field of data files.

assoctableid

A name or number used to identify the assoctable resource.

extension

3 letter file extension that is used to identify files of this type if they have no EA TYPE entry. (This may be empty.)

flags

EAF_DEFAULTOWNER

The default application for the file.

EAF_UNCHANGEABLE

This flag is set if the entry in the ASSOCTABLE is not to be edited.

EAF_REUSEICON

This flag is specified if a previously defined icon in the ASSOCTABLE is to be reused. Entries with this flag set have no icon data defined. The icon used for this entry is the icon used for the previous entry (see below). Note that EAF_* flags may be ORed together when specified in the ASSOCTABLE.

icon filename

Filename of the icon used to represent this file type. (This may be empty.)

Example

```
ASSOCTABLE  3000
BEGIN
"Product XYZ Spreadsheet", "xys", EAF_DEFAULTOWNER, xyzspr.ico
"Product XYZ Chart", "xyc", EAF_DEFAULTOWNER | EAF_REUSEICON
END
```

Dialog and Window Template Statements

This section describes how to define dialog and window templates.

It also describes the control data and presentation parameter structures that the application needs to create windows and define dialog templates.

DLGTEMPLATE and WINDOWTEMPLATE statements are used by an application to create predefined window and dialog resource templates. These statements are treated identically by the resource compiler and have the following format:

```
DLGTEMPLATE      resourceid
WINDOWTEMPLATE

loadoption      memoption      codepage
BEGIN          DIALOG statement      END
               CONTROL statement
               WINDOW statement
```

In the following description of the parts of the DLGTEMPLATE and WINDOWTEMPLATE statements, data types are shown after each parameter or option. These are the data types that the parameter or option is converted to when it is compiled.

Purpose

The DLGTEMPLATE or WINDOWTEMPLATE statement marks the beginning of a window template. It defines the name of the window, and its memory and load options.

resourceid (**USHORT**)

This is either:

- An integer (or a name that resolves to an integer) in the range of 1 through 65535
- A unique string enclosed in double quote marks

loadoption (**LOADOPTION**)

The default is LOADONCALL.

See [Resource Load and Memory Options](#) for a description of *LOADOPTION*.

memoption (**MEMOPTION**)

The default is MOVEABLE.

See [Resource Load and Memory Options](#) for a description of *MEMOPTION*.

code page (**USHORT**)

The code page of the text in the template.

Alternatively, {*i*} can be used in place of BEGIN and {*j*} in place of END.

The DLGTEMPLATE and WINDOWTEMPLATE keywords are synonymous.

The DIALOG statement defines a dialog-box window that can be created by an application and has the following format:

```
DIALOG    text , id , x , y , cx , cy

           ,style

           ,control

CTLDATA statement    PRESPARAMS statement

BEGIN      DIALOG statement      END
           CONTROL statement
```

WINDOW statement

[Control Data Statement](#)
[Presentation Parameters Statement](#)

The WINDOW and CONTROL statements have the format:

```
WINDOW    text, id, x, y, cx, cy, class
CONTROL

        ,style

        ,control
```

CTLDATA statement PRESPARAMS statement

```
BEGIN          DIALOG statement      END
                CONTROL statement
                WINDOW statement
```

[Control Data Statement](#)
[Presentation Parameters Statement](#)

Note: The WINDOW and CONTROL keywords are synonymous.

The DIALOG, CONTROL, and WINDOW statements between the BEGIN and END statements are defined as child windows. Presentation parameters always apply to the whole control. They cannot be changed for the individual items within the control.

Following is the description of the parameters for these statements.

Purpose

These statements mark the beginning of a window. They define the starting location on the display screen, its width, its height, and other details such as style.

Note: Not all values may be specified for each statement type. For details, see the call syntax diagrams.

text (PCH)

A string, enclosed in double quotes, that is displayed in the title-bar control, if it exists. To insert a double-quote character (") in the text, use two double-quote characters ("").

id (USHORT)

Item identifier.

x,y (SHORT)

Integer numbers specifying the x- and y-coordinates on the display screen of the lower left corner of the dialog. X and y are in dialog coordinates. The exact meaning of the coordinates depends on the style defined by the style argument. For normal dialogs, the coordinates are relative to the origin of the parent window. For FCF_SCREENALIGN style boxes, the coordinates are relative to the origin of the display screen. With FCF_MOUSEALIGN, the coordinates are relative to the position of the pointer at the time the dialog is created.

cx,cy (SHORT)

Integer numbers specifying the width and height of the window.

class (PCH)

The class of the window or control to be created.

Note: For a DIALOG statement the class is fixed as WC_FRAME and cannot be specified.

style (ULONG)

Any additional window style, frame style, or other class-specific style.

The default style is WS_SYNCPAINT | WS_CLIPSIBLINGS | WS_SAVEBITS | FS_DLGBOARDER. If the FS_DLGBOARDER or WS_SAVEBITS styles are not required, they should be preceded by the keyword "NOT". For example:

```
NOT FS_DLGBOARDER | FS_BORDER | NOT WS_SAVEBITS
```

replaces the `FS_DLGBCORDER` default style by the `FS_BORDER` style and removes the `WS_SAVEBITS` style. Note that the logic of the `NOT` keyword is different from the corresponding operator in the C language.

It is not possible to remove the default `WS_SYNCPAINT` and `WS_CLIPSIBLINGS` styles.

control (**ULONG**)

[Frame Creation Flags](#) (`FCF_*`) for the window.

This data is placed in the control data field in the correct format for a window of class `WC_FRAME`.

Note: `FCF_SHELLPOSITION` has no effect if specified in a template.

CTLDATA Statement

A statement used to define control data for the control. For more information on this statement, see [Control Data Statement](#)

PRESPARAMS Statement

A statement used to define presentation parameters. For more information on this statement, see [Presentation Parameters Statement](#)

MENU Statement

The `MENU` statement defines the contents of a menu resource. A menu resource is a collection of information that defines the appearance and function of an application menu. A menu can be used to create an action bar.

The `MENU` statement has the form:

```
MENU-menuid
    loadoption    memoption

    codepage

    PRESPARAMS statement
BEGIN
    MENUITEM statement    END
    SUBMENU statement
```

[Menu Item Statements](#)
[Submenu Statements](#)
[Presentation Parameters Statement](#).

menuid (**USHORT**)

This is either:

- An integer (or a name that resolves to an integer) in the range of 1 through 65535
- A unique string enclosed in double quote marks

loadoption (**LOADOPTION**)

The default is `LOADONCALL`.

See [Resource Load and Memory Options](#) for a description of *LOADOPTION*.

memoption (**MEMOPTION**)

The default is `MOVEABLE`.

See [Resource Load and Memory Options](#) for a description of *MEMOPTION*.

codepage (**USHORT**)

The code page of the text.

PRESPARAMS statement

A special resource statement used to define presentation parameters. These are discussed in more detail in [Presentation Parameters Statement](#).

MENUITEM statement

A special resource statement used to define the items in the menu. These are discussed in more detail in [Menu Item Statements](#).

SUBMENU statement

A special resource statement used to define a submen. SUBMENU statements are discussed in more detail in [Submenu Statements](#).

Example

Following is an example of a complete MENU statement:

```
MENU "sample"
BEGIN
  MENUITEM "~Alpha", 100, MIS_TEXT
  SUBMENU "~Beta", 101, MIS_TEXT
  BEGIN
    MENUITEM "~Green", 200, MIS_TEXT
    MENUITEM "~Blue", 201, MIS_TEXT, MIA_CHECKED
  END
END
END
```

Menu Item Statements

MENUITEM statements are used in the item-definition section of a MENU statement to define the names and attributes of the actual menu items. Any number of statements can be given; each defines a unique item. The order of the statements defines the order of the menu items.

Note: The MENUITEM statements can only be used within an item-definition section of a MENU statement.

```
MENUITEM
    string      , cmd      , styles      , attributes
                SEPARATOR
```

string (**PCH**)

A string, enclosed in double quotation marks, specifying the text of the menu item.

To insert a double-quote character (") in the text, use two double-quote characters ("").

If the **styles** parameter does not contain MIS_TEXT, the string is ignored but must still be specified. An empty string ("") should be specified in this instance.

To indicate the mnemonic for each item, insert the tilde character (~) in the string preceding the mnemonic character.

For MENUITEM statements within a SUBMENU (that is, pull-down menus) text may be split into a second column with an alignment substring. To right-align items insert "r" in the text where alignment should begin. To left-align a second column of text insert "l" in the text where alignment should begin. For each SUBMENU the longest item in the second column determines the width of that column. Only one alignment substring should be used in a menu item.

cmd (**USHORT**)

The value of the WM_COMMAND, WM_HELP, or WM_SYSCOMMAND message generated by the item when it is selected. It identifies the selection made and should be unique within one menu definition.

styles (**USHORT**)

One or more menu options defined by the MIS_* constants, ORed together with the "|" operator. For definitions of the MIS_*

constants, see [Menu Item Styles](#).

attributes (**USHORT**)

One or more menu options defined by the MIA_* constants, ORed together with the "|" operator. For definitions of the MIA_* constants, see [Menu Item Attributes](#).

The style MIS_SUBMENU must not be used with this statement. See [Submenu Statements](#) for the SUBMENU statement.

Examples:

```
MENUITEM  "Alpha", 1, MIS_TEXT,MIA_ENABLED|MIA_CHECKED,'A'
MENUITEM  "Beta", 2, MIS_TEXT,, 'B'
```

Submenu Statements

In addition to simple items, a menu definition can contain the definition of a submenu. A submenu can itself invoke a lower level submenu.

```
SUBMENU
    string      , cmd      , styles      , attributes      ,
    PRESPARAMS statement
BEGIN
    MENUITEM statement      END
    SUBMENU statement
```

string (**PCH**)

A string, enclosed in double quotation marks, specifying the text of the menu item.

To insert a double-quote character (") in the text, use two double-quote characters (").

If the **styles** parameter does not contain MIS_TEXT, the string is ignored but must still be specified. An empty string ("") should be specified in this instance.

cmd (**USHORT**)

The value of the WM_COMMAND, WM_HELP, or WM_SYSCOMMAND message generated by the item when it is selected. It identifies the selection made and should be unique within one menu definition.

styles (**USHORT**)

One or more menu options defined by the MIS_ constants, ORed together with the "|" operator.

In the SUBMENU statement, the style MIS_SUBMENU is always ORed with the styles given. If no value is supplied, the default value of MIS_TEXT and MIS_SUBMENU is used.

attributes (**USHORT**)

One or more menu options defined by the MIA_ constants, ORed together with the | operator.

Example:

```
MENU  "chem"
BEGIN

SUBMENU  "~Elements", 2, MIS_TEXT
BEGIN
    MENUITEM  "~Oxygen", 200, MIS_TEXT
    MENUITEM  "~Carbon", 201, MIS_TEXT,MIA_CHECKED
```

```

        MENUITEM "~Hydrogen", 202, MIS_TEXT
END

SUBMENU "~Compounds", 3, MIS_TEXT
BEGIN
    MENUITEM "~Glucose", 301, MIS_TEXT
    MENUITEM "~Sucrose", 302, MIS_TEXT,MIA_CHECKED
    MENUITEM "~Lactose", 303, MIS_TEXT|MIS_BREAK
    MENUITEM "~Fructose", 304, MIS_TEXT
END
END

```

SEPARATOR Menu Item

There is a special form of the MENUITEM statement that is used to create a horizontal dividing bar between two active menu items in a pull-down menu. The SEPARATOR menu item is itself inactive and has no text associated with it nor a **cmd** value.

Example

```

MENUITEM "~Roman", 206, MIS_TEXT
MENUITEM SEPARATOR
MENUITEM "20 ~Point", 301, MIS_TEXT

```

Menu Template

Menu templates are data structures used to define menus. Menu templates can be loaded as resources or created dynamically, or embedded in dialog templates, which in turn can be loaded as resources or created dynamically. Templates loaded as resources cannot contain references to bit maps or owner-drawn items. A menu template consists of a sequence of variable-length records. Each record in a menu template defines a menu item. If a menu item contains a reference to a submenu, the menu template that defines that submenu is placed after the definition of that particular menu item.

Template Format

A menu template has the following format:

Length (**USHORT**)

The length of the menu template.

Version (**USHORT**)

The template version. Versions 0 and 1 are valid.

Code page (**USHORT**)

The identifier of the code page used for the text items within the menu (but not any submenus, which each have their own code pages).

Item offset (**USHORT**)

The offset of the items from the start of the template, in bytes.

Count (**USHORT**)

The count of menu items.

Presentation parameters offset (**USHORT**)
Offset of presentation parameters from the start of the template, in bytes. This field is only present for version 1 of the template.

Menu Items
A variable-sized array of menu items as follows:

Style (**USHORT**)
Menu item styles (MIS_*; see [Menu Item Styles](#)) combined with the logical-OR operator.

Attributes (**USHORT**)
Menu item attributes (MIA_*; see [Menu Item Attributes](#)) combined with the logical-OR operator.

Item (**USHORT**)
An application-provided identifier for the menu item.

Variable data
Following the identifier is a variable data structure whose format depends upon the value of **Style**:

MIS_TEXT

Text (**PSZ**)
Null-terminated text string.

MIS_SUBMENU
A menu template structure.

MIS_BITMAP

Text (**PCH**)
Null-terminated text string.

For MIS_BITMAP menu items, the item text string can be used to derive the resource identifier from which a bit map is loaded. There are three instances:

- The first byte is null; that is, no resource is defined and it is assumed that the application subsequently provides a bit-map handle for the item.
- The first byte is 0xFF, the second byte is the low byte of the resource identifier, and the third byte is the high byte of the resource identifier.
- The first character is "#", and subsequent characters make up the decimal text representation of the resource identifier.

The resource is assumed to reside in the resource file of the current process.

If the string is empty or does not follow the format above, no resource is loaded.

STRINGTABLE Statement

The STRINGTABLE statement defines one or more string resources for an application. String resources are null-terminated ASCII strings that can be loaded, when needed, from the executable file, using the [WinLoadString](#) function.

Note: The ASCII strings can include no more than 256 characters, including the NULL termination character.

The STRINGTABLE statement has the form:

```
STRINGTABLE
    loadoption    memoption
    BEGIN    string-definitions    END    `
```

String-definitions

integer string ~

loadoption (LOADOPTION)
An optional keyword specifying when the resource is to be loaded. It must be one of:

PRELOAD	Resource is loaded immediately.
LOADONCALL	Resource is loaded when called.

The default is LOADONCALL.

See [Resource Load and Memory Options](#) for a description of *LOADOPTION*.

memoption (MEMOPTION)
Consists of the following keyword or keywords, specifying whether the resource is fixed or movable and whether it is discardable:

FIXED	Resource remains at a fixed memory location.
MOVEABLE	Resource can be moved if necessary to compact memory.
DISCARDABLE	Resource can be discarded if no longer needed.

The default is MOVEABLE and DISCARDABLE.

See [Resource Load and Memory Options](#) for a description of *MEMOPTION*.

string (PCH)
A string, enclosed in double quotation marks. To insert a double-quote character (") in the text, use two double-quote characters ("").

Note: A string may be defined on more than one line if each line begins and ends with a double-quote. If newline characters are desired after each line, there should be a double-quote at the beginning of the first line and at the end of the last line only. The string may contain any ASCII characters. Because (\) is interpreted as an escape character, use (\\) to generate a (\).

The following escape sequences may be used:

Escape Sequence	Name
\t	Horizontal tab
\a	Bell (alert)
\nnn	ASCII character (octal)
\xdd	ASCII character (hexadecimal).

The sequences \ddd and \xdd allow any character in the ASCII character set to be inserted in the character string. Thus, the horizontal tab could be entered as \X09, \011 or \t.

Example

```
#define IDS_STRING1 1
#define IDS_STRING2 2
#define IDS_STRING3 3

STRINGTABLE
BEGIN
    IDS_STRING1, "The first two strings in this table are identical."
    IDS_STRING2, "The first two strings "
                    "in this table are identical."
    IDS_STRING3, "This string will contain a newline character
                    before it continues on this line."
END
```

Templates, Control Data, and Presentation Parameters

This section describes:

- [Dialog Template](#)
- [Dialog Coordinates](#)
- [Dialog Template Format and Contents](#)
- [Header](#)
- [Items](#)
- [Data Area](#)
- [Control Data Statement](#)
- [Presentation Parameters Statement](#)
- [Parent/Child/Owner Relationship](#)
- [Predefined Window Classes](#)
- [Predefined Control Statements](#)

Dialog Template

A dialog template is a data structure used to define a dialog box. Dialog templates can be loaded from resources or created dynamically in memory. Dialog templates define windows of any window class that contain child windows of any class. For standard dialog windows, the dialog window itself is created with the `WC_FRAME` class, and its children are any of the preregistered control classes.

The dialog template specifies all the information required to create a dialog box and its children.

Dialog Coordinates

Coordinates in a dialog template are specified in *dialog coordinates*. These are based on the default character cell size; a unit in the horizontal direction is 1/4 the default character-cell width, and a unit in the vertical direction is 1/8 the default character-cell height. The origin is the bottom left-hand corner of the dialog box.

Dialog Template Format and Contents

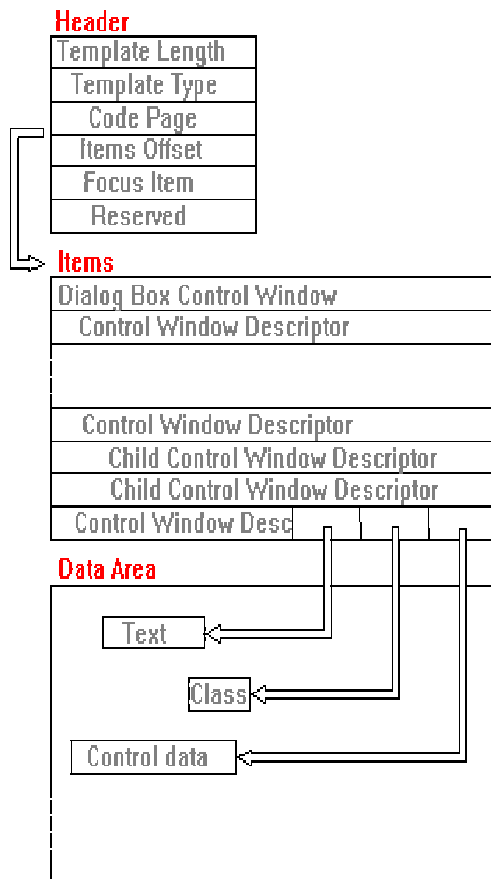
A dialog template has these sections:

Header	Defines the type of template format and contains information about the location of the other sections of the template. It also contains a summary of the status of the individual controls contained within the dialog box.
Items	Defines each of the controls that comprise the dialog box.
Data area	Contains the data values associated with each control. Each control defined in the item section contains pointers to the data area section. The data area also contains presentation parameter definitions. The data area is not necessarily a contiguous portion of the template. User data can be placed anywhere in the template if it does not interfere with other defined information.

The sections of a dialog template are illustrated in the following figure.

Notes

1. Throughout the dialog template all lengths are in bytes. String lengths do not include any null terminator that may be present. When strings are passed to the Presentation Interface, the length specifications are used and any null terminators are ignored. When strings are returned by the Presentation Interface, length specifications and null terminators are both supplied; therefore, space must be allowed for a null terminator.
2. All offsets are in bytes from the start of the dialog template structure.



Header

- The dialog template header consists of:
- Template length (USHORT)**
 The overall length of the dialog template.
 - Template type (USHORT)**
 The dialog template format type. The format defined is type 0.
 - Code page (USHORT)**
 The code page of the text in the dialog template.
 - Items offset (USHORT)**
 The offset of the array of dialog items.
 - Reserved (USHORT)**
 Must be 0.
 - Focus item (USHORT)**
 The index in the array of dialog items of the control to receive the focus. If this value is 0, or if the identified control cannot receive the focus, for example because it is a static control, the focus is passed to the first item within the template that can receive the focus.
 - Reserved (USHORT)**
 Must be 0.

Items

The dialog template items are specified as elements of an array that also defines the hierarchy of the control windows of the dialog box. Each element of the array is a control window descriptor and defines some control or a child of some control, so that every control within the dialog box is described by this array. The first descriptor is the specification of the dialog box itself.

The dialog template items consist of:

Reserved (**USHORT**) (**BOOL16**)
Must be 0.

Children (**USHORT**)
The number of dialog item child windows that are owned by this dialog item.

This is the number of elements following in the array that are created as child windows of this window. Each window can have any number of child windows, which allows for a tree-structured arrangement.

For example, in the figure in [Dialog Template Format and Contents](#), assuming that there are no more dialog items than are shown, the first item, the dialog box control window descriptor, has three children. The second item has no children, the third item has two children, and the remaining three items have no children.

Class name length (**USHORT**)
The length of the window class name string.

Class name offset (**USHORT**)
The offset of the window class name string.

Text length (**USHORT**)
The length of the text string.

For controls that allow input of text, this is the current text length, not the maximum text length, and so this value changes when text is put into the control.

Text offset (**USHORT**)
The offset of the text string.

Style (**ULONG**) (**BOOL32**)
The window style of the control.

The standard style bits are 16 bits. The use of the remaining 16 bits depends on the class of the control.

x (**SHORT**)

y (**SHORT**)
The position of the origin of the dialog item. This is specified in dialog coordinates, with x and y relative to the origin of the parent window.

cx (**SHORT**)

cy (**SHORT**)
The size of the dialog item in dialog coordinates; it must be greater than 0.

Identifier (**USHORT**)
An application-defined identifier for the dialog item.

Reserved (**USHORT**)
Must be zero.

Control data offset (**USHORT**)
The offset of the control-specific data for this dialog item. A value of 0 indicates that there is no control data for this dialog item.

Data Area

The dialog template data area contains the following different types of objects: **text**, **class name**, **presentation parameters**, and **control data**. These objects can be placed anywhere within the data area. They do not have to be in contiguous storage, and so an application can place data for its own use between these objects.

The dialog template data area contains:

Text (PCH)

The textual data associated with a dialog item.

Class name (PCH)

The name of the window class.

Presentation parameters (PRESPARAMS)

Presentation parameters are defined in [Presentation Parameters Statement](#).

Control data (CTLDATA)

For more information, see [Control Data Statement](#).

Control Data Statement

The optional CTLDATA statement is used to define control data for the control. Hexadecimal or decimal word constants follow the CTLDATA statement, separated with commas.

```
CTLDATA      decimal value
             hexadecimal value
             string
```

In addition to hexadecimal or decimal data, the CTLDATA statement can be followed by the MENU keyword, followed by a menu template in a BEGIN/END block. This creates a menu template as the control data of the window.

Presentation Parameters Statement

The optional PRESPARAMS statement is used to define presentation parameters. The syntax of the PRESPARAMS statement is as follows.

```
PRESPARAMS  type , value
```

A presentation parameter consists of:

type (ULONG)

The presentation parameter attribute type. See the [PARAM](#) data type for a description of valid types.

A string can be used to specify the type for a user type. If this is done, the string type is converted into a string atom when the dialog template is read into memory. Thereafter, this presentation parameter is referred to by this string atom. The application can use the atom manager API to match the string and the string atom.

value (LONG or PSZ)

One or more values depending upon the attribute type.

If the value is enclosed in quotes it is a zero-terminated string. Otherwise, it is converted to a LONG. There may be more than one value, depending upon the type. See [PARAM](#) data type for a description of the values required for system-defined presentation parameters.

Examples

The following are examples of PRESPARAMS statements:

```
PRESPARAMS PP_BORDERCOLOR, 0x00ff00ffL
PRESPARAMS PP_FONTNAMESIZE, "12.Helv"
PRESPARAMS "my color", 0x00ff00ffL
PRESPARAMS "my param", 0, 1, 2, 3, "Hi there"
```

Parent/Child/Owner Relationship

The format of the DLGTEMPLATE and WINDOWTEMPLATE resources is very general to allow tree-structured relationships within the resource format. The general layout of the templates is:

```
WINDOWTEMPLATE id
BEGIN
    WINDOW winTop           the top-level window
    BEGIN
        WINDOW wind1
        WINDOW wind2
        WINDOW wind3
        BEGIN
            WINDOW wind4
        END
        WINDOW wind5
    END
END
```

In this example, the top-level window is identified by **winTop**. It has four child windows: **wind1**, **wind2**, **wind3**, and **wind5**. **wind3** has one child window, **wind4**. When each of these windows is created, the parent and the owner are set to be the same.

The only time when the parent and owner windows are not the same is when frame controls are automatically created by a frame window.

Note that the WINDOW statements in the example above could also have been CONTROL or DIALOG statements.

Predefined Window Classes

The CONTROL statement can be used to define a window control of any class. Window classes may be user defined or one of a predefined set provided by the operating system. The following classes are provided in the OS/2 operating system.

WC_FRAME	Application frame control.
WC_STATIC	Text and group boxes.
WC_BUTTON	Push button, check box or radio button.
WC_COMBOBOX	Combination of an entry field and list box.
WC_ENTRYFIELD	Single line entry field.
WC_MLE	Multiple line entry field.
WC_LISTBOX	List box.
WC_MENU	Application action bar, menus and popup menus.
WC_SCROLLBAR	Horizontal or vertical scroll bar.
WC_TITLEBAR	Application title bar.
WC_SPINBUTTON	Spin button entry field.
WC_CONTAINER	Container list.
WC_SLIDER	Horizontal or vertical slider control.
WC_VALUESET	Value set control.
WC_NOTEBOOK	Notebook control.

These controls make up the standard user interface components for applications. The following example shows a simple listbox control.

```
CONTROL "", 1, 10, 20, 60, 40, WC_LISTBOX, WS_VISIBLE
```

Predefined Control Statements

In addition to the general form of the CONTROL statement, there are special control statements for commonly used controls. These statements define the attributes of the child control windows that appear in the window.

Control statements have this general form:

```
controltype text , id , x , y , cx , cy

, style

BEGIN      DIALOG statement      END
           CONTROL statement
           WINDOW statement
```

The following six controls are exceptions to this form because they do not take a text field. See the LISTBOX control statement for the form of these six controls.

- CONTAINER
- LISTBOX
- NOTEBOOK
- SLIDER
- SPINBUTTON
- VALUESET

controltype
is one of the keywords described below, defining the type of the control.

text (PCH)
is a string specifying the text to be displayed. The string must be enclosed in double quotation marks. The manner in which the text is displayed depends on the particular control, as detailed below.

To indicate the mnemonic for each item, insert the tilde character (~) in the string preceding the mnemonic character.

The double quotation marks are required for the COMBOBOX title even if no title is used.

id (USHORT)
is a unique integer number identifying the control.

x,y (SHORT)
are integer numbers specifying the x- and y-coordinates of the lower left corner of the control, in dialog coordinates. The coordinates are relative to the origin of the dialog.

cx,cy (SHORT)
are integer numbers specifying the width and height of the control.

The x, y, cx, and cy fields can use addition and subtraction operators (+ and -). For example, 15 + 6 can be used for the x-field.

Styles can be combined using the (|) operator.

The control type keywords are shown below, with their classes and default styles:

AUTOCHECKBOX

Class	WC_BUTTON
Default style	WS_TABSTOP, WS_VISIBLE, BS_AUTOCHECKBOX

AUTORADIOBUTTON

Class	WC_BUTTON
Default style	BS_AUTORADIOBUTTON, WS_TABSTOP, WS_VISIBLE

CHECKBOX

Class	WC_BUTTON
Default style	BS_CHECKBOX, WS_TABSTOP, WS_VISIBLE

COMBOBOX

Format	<p>The form of the COMBOBOX control statement is shown below.</p> <p>The fields have the same meaning as in the other control statements.</p>
--------	---

```
COMBOBOX "title" , id , x , y , cx
      , cy
      , style
```

Class	WC_COMBOBOX
Default style	CBS_SIMPLE, WS_TABSTOP, WS_VISIBLE

CONTAINER

Format	<p>The CONTAINER control statement does not contain a text field, so it has the same format as the LISTBOX statement.</p>
--------	---

Class	WC_CONTAINER
Default style	WS_TABSTOP, WS_VISIBLE, CCS_SINGLESEL

CTEXT

Class	WC_STATIC
Default style	SS_TEXT, DT_CENTER, WS_GROUP, WS_VISIBLE

DEFPUSHBUTTON

Class	WC_BUTTON
Default style	BS_DEFAULT, BS_PUSHBUTTON, WS_TABSTOP, WS_VISIBLE

EDITTEXT

Class	WC_ENTRYFIELD
Default style	WS_TABSTOP, WS_VISIBLE, ES_AUTOSCROLL

ENTRYFIELD

Class	WC_ENTRYFIELD
Default style	WS_TABSTOP, ES_LEFT, WS_VISIBLE

FRAME

Class	WC_FRAME
Default style	WS_VISIBLE

GROUPBOX

Class	WC_STATIC
Default style	SS_GROUPBOX, WS_TABSTOP, WS_VISIBLE

ICON

Class	WC_STATIC
Default style	SS_ICON, WS_VISIBLE
LISTBOX	
Format	The form of the LISTBOX control statement is different from the general form because it does not take a text field, however the fields have the same meaning as in the other control statements. The form of the LISTBOX control statement is shown below.
	<pre> controltype id , x , y , cx , cy , style </pre>
Class	WC_LISTBOX
Default style	WS_BORDER, WS_VISIBLE
LTEXT	
Class	WC_STATIC
Default style	SS_TEXT, DT_LEFT, WS_GROUP, WS_VISIBLE
MLE	
Class	WC_MLE
Default style	WS_TABSTOP, WS_VISIBLE, MLS_BORDER
NOTEBOOK	
Format	The NOTEBOOK control statement does not contain a text field, so it has the same format as the LISTBOX statement.
Class	WC_NOTEBOOK
Default style	WS_TABSTOP, WS_VISIBLE
PUSHBUTTON	
Class	WC_BUTTON
Default style	BS_PUSHBUTTON, WS_TABSTOP, WS_VISIBLE
RADIOBUTTON	
Class	WC_BUTTON
Default style	BS_RADIOBUTTON, WS_TABSTOP, WS_VISIBLE
RTEXT	
Class	WC_STATIC
Default style	SS_TEXT, DT_RIGHT, WS_GROUP, WS_VISIBLE
SLIDER	
Format	The SLIDER control statement does not contain a text field, so it has the same format as the LISTBOX statement.
Class	WC_SLIDER
Default style	WS_SLIDER, WS_TABSTOP, WS_VISIBLE
SPINBUTTON	
Format	The SPINBUTTON control statement does not contain a text field, so it has the same format as the LISTBOX statement.

Class	WC_SPINBUTTON
Default style	WS_TABSTOP, WS_VISIBLE, SPBS_MASTER

VALUESET

Format	The VALUESET control statement does not contain a text field, so it has the same format as the LISTBOX statement.
--------	---

Class	WC_VALUESET
Default style	WS_TABSTOP, WS_VISIBLE

Examples

The following is a complete example of a DIALOG statement:

```
DLGTEMPLATE "errmess"
BEGIN
    DIALOG "Disk Error", 100, 10, 10, 300, 110
    BEGIN
        CTEXT "Select One:", 1, 10, 80, 280, 12
        RADIOBUTTON "Retry", 2, 75, 50, 60, 12
        RADIOBUTTON "Abort", 3, 75, 30, 60, 12
        RADIOBUTTON "Ignore", 4, 75, 10, 60, 12
    END
END
```

This is an example of a WINDOWTEMPLATE statement that is used to define a specific kind of window frame. Calling Load Dialog with this resource automatically creates the frame window, the frame controls, and the client window (of class MyClientClass).

```
WINDOWTEMPLATE "wind1"
BEGIN
    FRAME "My Window", 1, 10, 10, 320, 130, WS_VISIBLE,
        FCF_STANDARD | FCF_VERTSCROLL
    BEGIN
        WINDOW "", FID_CLIENT, 0, 0, 0, 0, "MyClientClass",
            style
    END
END
```

This example creates a resource template for a parallel dialog identified by the constant **parallel1**. It includes a frame with a title bar, a system menu, and a dialog-style border. The parallel dialog has three auto radio buttons in it.

```
DLGTEMPLATE parallel1
BEGIN
    DIALOG "Parallel Dialog", 1, 50, 50, 180, 110
    CTLDATA FCF_TITLEBAR | FCF_SYSMENU | FCF_DLGBORDER
    BEGIN
        AUTORADIOBUTTON "Retry", 2, 75, 80, 60, 12
        AUTORADIOBUTTON "Abort", 3, 75, 50, 60, 12
        AUTORADIOBUTTON "Ignore", 4, 75, 30, 60, 12
    END
END
```

Resource (.RES) File Specification

The format for the .RES file is:

```
(/TYPE NAME FLAGS SIZE BYTES/)+
```

Where:

TYPE is either a null-terminated string or an ordinal, in which instance the first byte is 0xFF followed by an INT that is the ordinal.

```
/* Predefined resource types */
#define RT_POINTER 1
#define RT_BITMAP 2
#define RT_MENU 3
#define RT_DIALOG 4
#define RT_STRING 5
#define RT_FONTDIR 6
#define RT_FONT 7
#define RT_ACCELTABLE 8
#define RT_RCDATA 9
#define RT_DLGINCLUDE 11
#define RT_FKALONG 17
#define RT_HELPTABLE 18
#define RT_RMID 100
#define RT_RIFF 101
#define RT_WAVE 102
#define RT_AVI 103
#define RT_RESNAMES 255
```

NAME is the same format as TYPE. There are no predefined names.

FLAGS is an unsigned value containing the memory manager flags:

```
#define NSTYPE 0x0007 /* Segment type mask */
#define NSCODE 0x0000 /* Code segment */
#define NSDATA 0x0001 /* Data segment */
#define NSITER 0x0008 /* Iterated segment flag */
#define NSMOVE 0x0010 /* Moveable segment flag */
#define NSPURE 0x0020 /* Pure segment flag */
#define NSPRELOAD 0x0040 /* Preload segment flag */
#define NSEXRD 0x0080 /* Execute-only (code segment),
/* or read-only (data segment) */
#define NSRELOC 0x0100 /* Segment has relocations */
#define NSCONFORM 0x0200 /* Segment has debug info */
#define NSDPL 0x0C00 /* 286 DPL bits */
#define NSDISCARD 0x1000 /* Discard bit for segment */
#define NS32BIT 0x2000 /* 32-BIT code segment */
#define NSHUGE 0x4000 /* Huge memory segment */
```

SIZE is a LONG value defining how many bytes follow in the resource.

BYTES is the stream of bytes that makes up the resource.

Any number of resources can appear one after another in the .RES file.

Note: For descriptions and examples of the multimedia resource types of MIDI, RIFF, WAVE digital audio, and AVI digital video, see the *OS/2 Warp Multimedia Programming Reference*.

Virtual Key Definitions

The PC VKEY set is shown in the following table:

Symbol	Personal Computer AT Keyboard	Enhanced Keyboard
--------	-------------------------------	-------------------

VK_BUTTON1 VK_BUTTON2 VK_BUTTON3	These values are only used to access the up/down and toggled states of the pointing device buttons; they never actually appear in a WM_CHAR message.	These values are only used to access the up/down and toggled states of the pointing device buttons; they never actually appear in a WM_CHAR message.
VK_BREAK	Ctrl + Scroll Lock	Ctrl + Pause
VK_BACKSPACE	Backspace	Backspace
VK_TAB	Tab	Tab
VK_BACKTAB	Shift + Tab	Shift + Tab
VK_NEWLINE	Enter	Enter
VK_SHIFT *	Left and Right Shift	Left and Right Shift
VK_CTRL *	Ctrl	Left and Right Ctrl
VK_ALT *	Alt	Left and Right Alt
VK_ALTFGRAF *	None	Alt Graf (if available)
VK_PAUSE	Ctrl + Num Lock	Pause
VK_CAPSLOCK	Caps Lock	Caps Lock
VK_ESC	Esc	Esc
VK_SPACE *	Space	Space
VK_PAGEUP *	Numpad 9	Pg Up and Numpad 9
VK_PAGEDOWN *	Numpad 3	Pg Dn and Numpad 3
VK_END *	Numpad 1	End and Numpad 1
VK_HOME *	Numpad 7	Home and Numpad 7
VK_LEFT *	Numpad 4	Left and Numpad 4
VK_UP *	Numpad 8	Up and Numpad 8
VK_RIGHT *	Numpad 6	Right and Numpad 6
VK_DOWN *	Numpad 2	Down and Numpad 2
VK_PRINTSCRN	Shift + Print Screen	Print Screen
VK_INSERT *	Numpad 0	Ins and Numpad 0
VK_DELETE *	Numpad	Del and Numpad
VK_SCROLLLOCK	Scroll Lock	Scroll Lock
VK_NUMLOCK	Num Lock	Num Lock
VK_ENTER	Shift + Enter	Shift + Enter and Numpad Enter
VK_SYSRQ	SysRq	Alt + Print Screen
VK_F1 *	F1	F1
VK_F2 *	F2	F2
VK_F3 *	F3	F3
VK_F4 *	F4	F4
VK_F5 *	F5	F5
VK_F6 *	F6	F6
VK_F7 *	F7	F7

VK_F8	*	F8	F8
VK_F9	*	F9	F9
VK_F10	*	F10	F10
VK_F11	*	None	F11
VK_F12	*	None	F12
VK_F13		None	None
VK_F14		None	None
VK_F15		None	None
VK_F16		None	None
VK_F17		None	None
VK_F18		None	None
VK_F19		None	None
VK_F20		None	None
VK_F21		None	None
VK_F22		None	None
VK_F23		None	None
VK_F24		None	None
VK_MENU	*	F10	F10

Notes

1. VKEYs marked with an asterisk (*) are generated irrespective of other shift states (Shift, Ctrl, Alt, and Alt Graf).
2. VK_CAPSLOCK is **not** generated for any of the Ctrl shift states, for PC-DOS compatibility.
3. Wherever possible, the VK_ name is derived from the legend on the key top of the 101-key Enhanced PC keyboard.

Notices

November 1996

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Requests for technical information about IBM products should be made to your IBM reseller or IBM marketing representative.

Copyright Notices

COPYRIGHT LICENSE: This publication contains printed sample application programs in source language, which illustrate OS/2 programming techniques. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the OS/2 application programming interface.

Each copy of any portion of these sample programs or any derivative work, which is distributed to others, must include a copyright notice as follows: "(C) (your company name) (year). All rights reserved."

(C) Copyright International Business Machines Corporation 1994, 1996. All rights reserved.

Note to U.S. Government Users - Documentation related to restricted rights - Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Disclaimers

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

Asia-Pacific users can inquire, in writing, to the IBM Director of Intellectual Property and Licensing, IBM World Trade Asia Corporation, 2-31 Roppongi 3-chome, Minato-ku, Tokyo 106, Japan.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Department LZKS, 11400 Burnet Road, Austin, TX 78758 U.S.A. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

Common User Access
IBM
IPF
OS/2
Presentation Manager
System Application Architecture

CUA
Information Presentation Facility
Operating System/2
PM
SAA
Workplace Shell

The following terms are trademarks of other companies:

Adobe
Helvetica
Intel
PostScript
Times New Roman

Adobe Systems Incorporated
Linotype
Intel Corporation
Adobe Systems Incorporated
Monotype

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of

others.

Glossary

This glossary defines many of the terms used in this book. It includes terms and definitions from the *IBM Dictionary of Computing*, as well as terms specific to the OS/2 operating system and the Presentation Manager. It is not a complete glossary for the entire OS/2 operating system; nor is it a complete dictionary of computer terms.

Other primary sources for these definitions are:

- The *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyrighted 1990 by the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. These definitions are identified by the symbol (A) after the definition.
- The *Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

Glossary Listing

Select a starting letter of glossary terms:

A	N
B	O
C	P
D	Q
E	R
F	S
G	T
H	U
I	V
J	W
K	X
L	Y
M	Z

Glossary - A

accelerator - In SAA Common User Access architecture, a key or combination of keys that invokes an application-defined function.

accelerator table - A table used to define which key strokes are treated as *accelerators* and the commands they are translated into.

access mode - The manner in which an application gains access to a file it has opened. Examples of access modes are read-only, write-only, and read/write.

access permission - All access rights that a user has regarding an object. (I)

action - One of a set of defined tasks that a computer performs. Users request the application to perform an action in several ways, such as typing a command, pressing a function key, or selecting the action name from an action bar or menu.

action bar - In SAA Common User Access architecture, the area at the top of a window that contains choices that give a user access to actions available in that window.

action point - The current position on the screen at which the pointer is pointing. Contrast with *hot spot* and *input focus*.

active program - A program currently running on the computer. An active program can be interactive (running and receiving input from the user) or noninteractive (running but not receiving input from the user). See also *interactive program* and *noninteractive program*.

active window - The window with which the user is currently interacting.

address space - (1) The range of addresses available to a program. (A) (2) The area of virtual storage available for a particular job.

alphanumeric video output - Output to the logical video buffer when the video adapter is in text mode and the logical video buffer is addressed by an application as a rectangular array of character cells.

American National Standard Code for Information Interchange - The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), that is used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters. (A)

Note: IBM has defined an extension to ASCII code (characters 128-255).

anchor - A window procedure that handles Presentation Manager message conversions between an icon procedure and an application.

anchor block - An area of Presentation-Manager-internal resources to allocated process or thread that calls WinInitialize.

anchor point - A point in a window used by a program designer or by a window manager to position a subsequently appearing window.

ANSI - American National Standards Institute.

APA - All points addressable.

API - Application programming interface.

application - A collection of software components used to perform specific types of work on a computer; for example, a payroll application, an airline reservation application, a network application.

application object - In SAA Advanced Common User Access architecture, a form that an application provides for a user; for example, a spreadsheet form. Contrast with *user object*.

application programming interface (API) - A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program.

application-modal - Pertaining to a message box or dialog box for which processing must be completed before further interaction with any other window owned by the same application may take place.

area - In computer graphics, a filled shape such as a solid rectangle.

ASCII - American National Standard Code for Information Interchange.

ASCIIZ - A string of ASCII characters that is terminated with a byte containing the value 0.

aspect ratio - In computer graphics, the width-to-height ratio of an area, symbol, or shape.

asynchronous (ASYNCR) - (1) Pertaining to two or more processes that do not depend upon the occurrence of specific events such as common timing signals. (T) (2) Without regular time relationship; unexpected or unpredictable with respect to the execution of program instructions. See also *synchronous*.

atom - A constant that represents a string. As soon as a string has been defined as an atom, the atom can be used in place of the string to save space. Strings are associated with their respective atoms in an *atom table*. See also *integer atom*.

atom table - A table used to relate *atoms* with the strings that they represent. Also in the table is the mechanism by which the presence of a string can be checked.

atomic operation - An operation that completes its work on an object before another operation can be performed on the same object.

attribute - A characteristic or property that can be controlled, usually to obtain a required appearance; for example, the color of a line. See also *graphics attributes* and *segment attributes*.

automatic link - In Information Presentation Facility (IPF), a link that begins a chain reaction at the primary window. When the user selects the primary window, an automatic link is activated to display secondary windows.

AVIO - Advanced Video Input/Output.

Glossary - B

Bézier curve - (1) A mathematical technique of specifying smooth continuous lines and surfaces, which require a starting point and a finishing point with several intermediate points that influence or control the path of the linking curve. Named after Dr. P. Bézier. (2) (D of C) In the AIX Graphics Library, a cubic spline approximation to a set of four control points that passes through the first and fourth control points and that has a continuous slope where two spline segments meet. Named after Dr. P. Bézier.

background - (1) In multiprogramming, the conditions under which low-priority programs are executed. Contrast with *foreground*. (2) An active session that is not currently displayed on the screen.

background color - The color in which the background of a graphic primitive is drawn.

background mix - An attribute that determines how the background of a graphic primitive is combined with the existing color of the graphics presentation space. Contrast with *mix*.

background program - In multiprogramming, a program that executes with a low priority. Contrast with *foreground program*.

bit map - A representation in memory of the data displayed on an APA device, usually the screen.

block - (1) A string of data elements recorded or transmitted as a unit. The elements may be characters, words, or logical records. (T) (2) To record data in a block. (3) A collection of contiguous records recorded as a unit. Blocks are separated by interblock gaps and each block may contain one or more records. (A)

block device - A storage device that performs I/O operations on blocks of data called *sectors*. Data on block devices can be randomly accessed. Block devices are designated by a drive letter (for example, **C:**).

blocking mode - A condition set by an application that determines when its threads might block. For example, an application might set the Pipemode parameter for the DosCreateNPipe function so that its threads perform I/O operations to the named pipe block when no data is available.

border - A visual indication (for example, a separator line or a background color) of the boundaries of a window.

boundary determination - An operation used to compute the size of the smallest rectangle that encloses a graphics object on the screen.

breakpoint - (1) A point in a computer program where execution may be halted. A breakpoint is usually at the beginning of an instruction where halts, caused by external intervention, are convenient for resuming execution. (T) (2) A place in a program, specified by a command or a condition, where the system halts execution and gives control to the workstation user or to a specified program.

broken pipe - When all of the handles that access one end of a pipe have been closed.

bucket - One or more fields in which the result of an operation is kept.

buffer - (1) A portion of storage used to hold input or output data temporarily. (2) To allocate and schedule the use of buffers. (A)

button - A mechanism used to request or initiate an action. See also *barrel buttons*, *bezel buttons*, *mouse button*, *push button*, and *radio button*.

byte pipe - Pipes that handle data as byte streams. All unnamed pipes are byte pipes. Named pipes can be byte pipes or message pipes. See *byte stream*.

byte stream - Data that consists of an unbroken stream of bytes.

Glossary - C

cache - A high-speed buffer storage that contains frequently accessed instructions and data; it is used to reduce access time.

cached micro presentation space - A presentation space from a Presentation-Manager-owned store of micro presentation spaces. It can be used for drawing to a window only, and must be returned to the store when the task is complete.

CAD - Computer-Aided Design.

call - (1) The action of bringing a computer program, a routine, or a subroutine into effect, usually by specifying the entry conditions and jumping to an entry point. (I) (A) (2) To transfer control to a procedure, program, routine, or subroutine.

calling sequence - A sequence of instructions together with any associated data necessary to execute a call. (T)

Cancel - An action that removes the current window or menu without processing it, and returns the previous window.

cascaded menu - In the OS/2 operating system, a menu that appears when the arrow to the right of a cascading choice is selected. It contains a set of choices that are related to the cascading choice. Cascaded menus are used to reduce the length of a menu. See also *cascading choice*.

cascading choice - In SAA Common User Access architecture, a choice in a menu that, when selected, produces a cascaded menu containing other choices. An arrow () appears to the right of the cascading choice.

CASE statement - In PM programming, provides the body of a window procedure. There is usually one CASE statement for each message type supported by an application.

CGA - Color graphics adapter.

chained list - A list in which the data elements may be dispersed but in which each data element contains information for locating the next. (T) Synonymous with *linked list*.

character - A letter, digit, or other symbol.

character box - In computer graphics, the boundary that defines, in world coordinates, the horizontal and vertical space occupied by a single character from a character set. See also *character mode*. Contrast with *character cell*.

character cell - The physical, rectangular space in which any single character is displayed on a screen or printer device. Position is addressed by row and column coordinates. Contrast with *character box*.

character code - The means of addressing a character in a character set, sometimes called *code point*.

character device - A device that performs I/O operations on one character at a time. Because character devices view data as a stream of bytes, character-device data cannot be randomly accessed. Character devices include the keyboard, mouse, and printer, and are referred to by name.

character mode - A mode that, in conjunction with the font type, determines the extent to which graphics characters are affected by the character box, shear, and angle attributes.

character set - (1) An ordered set of unique representations called characters; for example, the 26 letters of English alphabet, Boolean 0 and 1, the set of symbols in the Morse code, and the 128 ASCII characters. (A) (2) All the valid characters for a programming language or for a computer system. (3) A group of characters used for a specific reason; for example, the set of characters a printer can print.

check box - In SAA Advanced Common User Access architecture, a square box with associated text that represents a choice. When a user selects a choice, an X appears in the check box to indicate that the choice is in effect. The user can clear the check box by selecting the choice again. Contrast with *radio button*.

check mark - (1) (D of C) In SAA Advanced Common User Access architecture, a symbol that shows that a choice is currently in effect. (2) The symbol that is used to indicate a selected item on a pull-down menu.

child process - In the OS/2 operating system, a process started by another process, which is called the parent process. Contrast with *parent process*.

child window - A window that appears within the border of its parent window (either a primary window or another child window). When the parent window is resized, moved, or destroyed, the child window also is resized, moved, or destroyed; however, the child window can be moved or resized independently from the parent window, within the boundaries of the parent window. Contrast with *parent window*.

choice - (1) An option that can be selected. The choice can be presented as text, as a symbol (number or letter), or as an icon (a pictorial symbol). (2) (D of C) In SAA Common User Access architecture, an item that a user can select.

chord - (1) To press more than one button on a pointing device while the pointer is within the limits that the user has specified for the operating environment. (2) (D of C) In graphics, a short line segment whose end points lie on a circle. Chords are a means for producing a circular image from straight lines. The higher the number of chords per circle, the smoother the circular image.

class - A way of categorizing objects based on their behavior and shape. A class is, in effect, a definition of a generic object. In SOM, a class is a special kind of object that can manufacture other objects that all have a common shape and exhibit similar behavior (more precisely, all of the objects manufactured by a class have the same memory layout and share a common set of methods). New classes can be defined in terms of existing classes through a technique known as *inheritance*.

class method - A class method of class <X> is a method provided by the metaclass of class <X>. Class methods are executed without

requiring any instances of class <X> to exist, and are frequently used to create instances. In System Object Model, an action that can be performed on a class object.

class object - In System Object Model, the run-time implementation of a class.

class style - The set of properties that apply to every window in a window class.

client - (1) A functional unit that receives shared services from a server. (T) (2) A user, as in a client process that uses a named pipe or queue that is created and owned by a server process.

client area - The part of the window, inside the border, that is below the menu bar. It is the user's work space, where a user types information and selects choices from selection fields. In primary windows, it is where an application programmer presents the objects that a user works on.

client program - An application that creates and manipulates instances of classes.

client window - The window in which the application displays output and receives input. This window is located inside the frame window, under the window title bar and any menu bar, and within any scroll bars.

clip limits - The area of the paper that can be reached by a printer or plotter.

clipboard - In SAA Common User Access architecture, an area of computer memory, or storage, that temporarily holds data. Data in the clipboard is available to other applications.

clipping - In computer graphics, removing those parts of a display image that lie outside a given boundary. (I) (A)

clipping area - The area in which the window can paint.

clipping path - A clipping boundary in world-coordinate space.

clock tick - The minimum unit of time that the system tracks. If the system timer currently counts at a rate of X Hz, the system tracks the time every 1/X of a second. Also known as *time tick*.

CLOCK\$ - Character-device name reserved for the system clock.

code page - An assignment of graphic characters and control-function meanings to all code points.

code point - (1) Synonym for *character code*. (2) (D of C) A 1-byte code representing one of 256 potential characters.

code segment - An executable section of programming code within a load module.

color dithering - See *dithering*.

color graphics adapter (CGA) - An adapter that simultaneously provides four colors and is supported by all IBM Personal Computer and Personal System/2 models.

command - The name and parameters associated with an action that a program can perform.

command area - An area composed of a command field prompt and a command entry field.

command entry field - An entry field in which users type commands.

command line - On a display screen, a display line, sometimes at the bottom of the screen, in which only commands can be entered.

command mode - A state of a system or device in which the user can enter commands.

command prompt - A field prompt showing the location of the command entry field in a panel.

Common Programming Interface (CPI) - Definitions of those application development languages and services that have, or are intended to have, implementations on and a high degree of commonality across the SAA environments. One of the three SAA architectural areas. See also *Common User Access architecture*.

Common User Access (CUA) architecture - Guidelines for the dialog between a human and a workstation or terminal. One of the three SAA architectural areas. See also *Common Programming Interface*.

compile - To translate a program written in a higher-level programming language into a machine language program.

composite window - A window composed of other windows (such as a frame window, frame-control windows, and a client window) that are kept together as a unit and that interact with each other.

computer-aided design (CAD) - The use of a computer to design or change a product, tool, or machine, such as using a computer for drafting or illustrating.

COM1, COM2, COM3 - Character-device names reserved for serial ports 1 through 3.

CON - Character-device name reserved for the console keyboard and screen.

conditional cascaded menu - A pull-down menu associated with a menu item that has a cascade mini-push button beside it in an object's pop-up menu. The conditional cascaded menu is displayed when the user selects the mini-push button.

container - In SAA Common User Access architecture, an object that holds other objects. A folder is an example of a container object. See also *folder* and *object*.

contextual help - In SAA Common User Access Architecture, help that gives specific information about the item the cursor is on. The help is contextual because it provides information about a specific item as it is currently being used. Contrast with *extended help*.

contiguous - Touching or joining at a common edge or boundary, for example, an unbroken consecutive series of storage locations.

control - In SAA Advanced Common User Access architecture, a component of the user interface that allows a user to select choices or type information; for example, a check box, an entry field, a radio button.

control area - A storage area used by a computer program to hold control information. (I) (A)

Control Panel - In the Presentation Manager, a program used to set up user preferences that act globally across the system.

Control Program - (1) The basic functions of the operating system, including DOS emulation and the support for keyboard, mouse, and video input/output. (2) A computer program designed to schedule and to supervise the execution of programs of a computer system. (I) (A)

control window - A window that is used as part of a composite window to perform simple input and output tasks. Radio buttons and check boxes are examples.

control word - An instruction within a document that identifies its parts or indicates how to format the document.

coordinate space - A two-dimensional set of points used to generate output on a video display or printer.

Copy - A choice that places onto the clipboard, a copy of what the user has selected. See also *Cut* and *Paste*.

correlation - The action of determining which element or object within a picture is at a given position on the display. This follows a *pick* operation.

coverpage window - A window in which the application's help information is displayed.

CPI - Common Programming Interface.

critical extended attribute - An extended attribute that is necessary for the correct operation of the system or a particular application.

critical section - (1) In programming languages, a part of an asynchronous procedure that cannot be executed simultaneously with a certain part of another asynchronous procedure. (I)

Note: Part of the other asynchronous procedure also is a critical section. (2) A section of code that is not reentrant; that is, code that can be executed by only one thread at a time.

CUA architecture - Common User Access architecture.

current position - In computer graphics, the position, in user coordinates, that becomes the starting point for the next graphics routine, if that routine does not explicitly specify a starting point.

cursor - A symbol displayed on the screen and associated with an input device. The cursor indicates where input from the device will be placed. Types of cursors include text cursors, graphics cursors, and selection cursors. Contrast with *pointer* and *input focus*.

Cut - In SAA Common User Access architecture, a choice that removes a selected object, or a part of an object, to the clipboard, usually compressing the space it occupied in a window. See also *Copy* and *Paste*.

Glossary - D

daisy chain - A method of device interconnection for determining interrupt priority by connecting the interrupt sources serially.

data segment - A nonexecutable section of a program module; that is, a section of a program that contains data definitions.

data structure - The syntactic structure of symbolic expressions and their storage-allocation characteristics. (T)

data transfer - The movement of data from one object to another by way of the clipboard or by direct manipulation.

DBCS - Double-byte character set.

DDE - Dynamic data exchange.

deadlock - (1) Unresolved contention for the use of a resource. (2) An error condition in which processing cannot continue because each of two elements of the process is waiting for an action by, or a response from, the other. (3) An impasse that occurs when multiple processes are waiting for the availability of a resource that will not become available because it is being held by another process that is in a similar wait state.

debug - To detect, diagnose, and eliminate errors in programs. (T)

decipoint - In printing, one tenth of a point. There are 72 points in an inch.

default procedure - A function provided by the Presentation Manager Interface that may be used to process standard messages from dialogs or windows.

default value - A value assumed when no value has been specified. Synonymous with assumed value. For example, in the graphics programming interface, the default line-type is 'solid'.

definition list - A type of list that pairs a term and its description.

delta - An application-defined threshold, or number of container items, from either end of the list.

descendant - See *child process*.

descriptive text - Text used in addition to a field prompt to give more information about a field.

Deselect all - A choice that cancels the selection of all of the objects that have been selected in that window.

Desktop Manager - In the Presentation Manager, a window that displays a list of groups of programs, each of which can be started or stopped.

desktop window - The window, corresponding to the physical device, against which all other types of windows are established.

detached process - A background process that runs independent of the parent process.

detent - A point on a slider that represents an exact value to which a user can move the slider arm.

device context - A logical description of a data destination such as memory, metafile, display, printer, or plotter. See also *direct device context*, *information device context*, *memory device context*, *metafile device context*, *queued device context*, and *screen device context*.

device driver - A file that contains the code needed to attach and use a device such as a display, printer, or plotter.

device space - (1) Coordinate space in which graphics are assembled after all GPI transformations have been applied. Device space is defined in device-specific units. (2) (D of C) In computer graphics, a space defined by the complete set of addressable points of a display device. (A)

dialog - The interchange of information between a computer and its user through a sequence of requests by the user and the presentation of responses by the computer.

dialog box - In SAA Advanced Common User Access architecture, a movable window, fixed in size, containing controls that a user uses to provide information required by an application so that it can continue to process a user request. See also *message box*, *primary window*, *secondary window*. Also known as a *pop-up window*.

Dialog Box Editor - A *WYSIWYG* editor that creates dialog boxes for communicating with the application user.

dialog item - A component (for example, a menu or a button) of a dialog box. Dialog items are also used when creating dialog templates.

dialog procedure - A dialog window that is controlled by a window procedure. It is responsible for responding to all messages sent to the dialog window.

dialog tag language - A markup language used by the DTL compiler to create dialog objects.

dialog template - The definition of a dialog box, which contains details of its position, appearance, and window ID, and the window ID of each of its child windows.

direct device context - A logical description of a data destination that is a device other than the screen (for example, a printer or plotter), and where the output is not to go through the spooler. Its purpose is to satisfy queries. See also *device context*.

direct manipulation - The user's ability to interact with an object by using the mouse, typically by dragging an object around on the Desktop and dropping it on other objects.

direct memory access (DMA) - A technique for moving data directly between main storage and peripheral equipment without requiring processing of the data by the processing unit.(T)

directory - A type of file containing the names and controlling information for other files or other directories.

display point - Synonym for *pel*.

dithering - (1) The process used in color displays whereby every other pel is set to one color, and the intermediate pels are set to another. Together they produce the effect of a third color at normal viewing distances. This process can only be used on solid areas of color; it does not work, for example, on narrow lines. (2) (D of C) In computer graphics, a technique of interleaving dark and light pixels so that the resulting image looks smoothly shaded when viewed from a distance.

DMA - Direct memory access.

DOS Protect Mode Interface (DPMI) - An interface between protect mode and real mode programs.

double-byte character set (DBCS) - A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more characters than can be represented by 256 code points, require double-byte character sets. Since each character requires two bytes, the entering, displaying, and printing of DBCS characters requires hardware and software that can support DBCS.

doubleword - A contiguous sequence of bits or characters that comprises two computer words and is capable of being addressed as a unit.
(A)

DPMI - DOS Protect Mode Interface.

drag - In SAA Common User Access, to use a pointing device to move an object; for example, clicking on a window border, and dragging it to make the window larger.

dragging - (1) In computer graphics, moving an object on the display screen as if it were attached to the pointer. (2) (D of C) In computer graphics, moving one or more segments on a display surface by translating. (I) (A)

drawing chain - See *segment chain*.

drop - To fix the position of an object that is being dragged, by releasing the select button of the pointing device. See also *drag*.

DTL - Dialog tag language.

dual-boot function - A feature of the OS/2 operating system that allows the user to start DOS from within the operating system, or an OS/2 session from within DOS.

duplex - Pertaining to communication in which data can be sent and received at the same time. Synonymous with *full duplex*.

dynamic data exchange (DDE) - A message protocol used to communicate between applications that share data. The protocol uses shared memory as the means of exchanging data between applications.

dynamic data formatting - A formatting procedure that enables you to incorporate text, bit maps or metafiles in an IPF window at execution time.

dynamic link library - A collection of executable programming code and data that is bound to an application at load time or run time, rather than during linking. The programming code and data in a dynamic link library can be shared by several applications simultaneously.

dynamic linking - The process of resolving external references in a program module at load time or run time rather than during linking.

dynamic segments - Graphics segments drawn in exclusive-OR mix mode so that they can be moved from one screen position to another without affecting the rest of the displayed picture.

dynamic storage - (1) A device that stores data in a manner that permits the data to move or vary with time such that the specified data is not always available for recovery. (A) (2) A storage in which the cells require repetitive application of control signals in order to retain stored data. Such repetitive application of the control signals is called a refresh operation. A dynamic storage may use static addressing or sensing circuits. (A) (3) See also *static storage*.

dynamic time slicing - Varies the size of the time slice depending on system load and paging activity.

dynamic-link module - A module that is linked at load time or run time.

Glossary - E

EBCDIC - Extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded characters (9 bits including parity check), used for information interchange among data processing systems, data communications systems, and associated equipment.

edge-triggered - Pertaining to an event semaphore that is posted then reset before a waiting thread gets a chance to run. The semaphore is considered to be posted for the rest of that thread's waiting period; the thread does not have to wait for the semaphore to be posted again.

EGA - Extended graphics adapter.

element - An entry in a graphics segment that comprises one or more graphics orders and that is addressed by the element pointer.

EMS - Expanded Memory Specification.

encapsulation - Hiding an object's implementation, that is, its private, internal data and methods. Private variables and methods are accessible only to the object that contains them.

entry field - In SAA Common User Access architecture, an area where a user types information. Its boundaries are usually indicated. See also *selection field*.

entry panel - A defined panel type containing one or more entry fields and protected information such as headings, prompts, and explanatory text.

entry-field control - The component of a user interface that provides the means by which the application receives data entered by the user in an entry field. When it has the input focus, the entry field displays a flashing pointer at the position where the next typed character will go.

environment segment - The list of environment variables and their values for a process.

environment strings - ASCII text strings that define the value of environment variables.

environment variables - Variables that describe the execution environment of a process. These variables are named by the operating system or by the application. Environment variables named by the operating system are PATH, DPATH, INCLUDE, INIT, LIB, PROMPT, and TEMP. The values of environment variables are defined by the user in the CONFIG.SYS file, or by using the SET command at the OS/2 command prompt.

error message - An indication that an error has been detected. (A)

event semaphore - A semaphore that enables a thread to signal a waiting thread or threads that an event has occurred or that a task has been completed. The waiting threads can then perform an action that is dependent on the completion of the signaled event.

exception - An abnormal condition such as an I/O error encountered in processing a data set or a file.

exclusive system semaphore - A system semaphore that can be modified only by threads within the same process.

executable file - (1) A file that contains programs or commands that perform operations or actions to be taken. (2) A collection of related data records that execute programs.

exit - To execute an instruction within a portion of a computer program in order to terminate the execution of that portion. Such portions of computer programs include loops, subroutines, modules, and so on. (T) Repeated exit requests return the user to the point from which all functions provided to the system are accessible. Contrast with *cancel*.

expanded memory specification (EMS) - Enables DOS applications to access memory above the 1MB real mode addressing limit.

extended attribute - An additional piece of information about a file object, such as its data format or category. It consists of a name and a value. A file object may have more than one extended attribute associated with it.

extended help - In SAA Common User Access architecture, a help action that provides information about the contents of the application window from which a user requested help. Contrast with *contextual help*.

extended-choice selection - A mode that allows the user to select more than one item from a window. Not all windows allow extended choice selection. Contrast with *multiple-choice selection*.

extent - Continuous space on a disk or diskette that is occupied by or reserved for a particular data set, data space, or file.

external link - In Information Presentation Facility, a link that connects external online document files.

Glossary - F

family-mode application - An application program that can run in the OS/2 environment and in the DOS environment; however, it cannot take advantage of many of the OS/2-mode facilities, such as multitasking, interprocess communication, and dynamic linking.

FAT - File allocation table.

FEA - Full extended attribute.

field-level help - Information specific to the field on which the cursor is positioned. This help function is "contextual" because it provides information about a specific item as it is currently used; the information is dependent upon the context within the work session.

FIFO - First-in-first-out. (A)

file - A named set of records stored or processed as a unit. (T)

file allocation table (FAT) - In IBM personal computers, a table used by the operating system to allocate space on a disk for a file, and to locate and chain together parts of the file that may be scattered on different sectors so that the file can be used in a random or sequential manner.

file attribute - Any of the attributes that describe the characteristics of a file.

File Manager - In the Presentation Manager, a program that displays directories and files, and allows various actions on them.

file specification - The full identifier for a file, which includes its drive designation, path, file name, and extension.

file system - The combination of software and hardware that supports storing information on a storage device.

file system driver (FSD) - A program that manages file I/O and controls the format of information on the storage media.

fillet - A curve that is tangential to the end points of two adjoining lines. See also *polyfillet*.

filtering - An application process that changes the order of data in a queue.

first-in-first-out (FIFO) - A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time. (A)

flag - (1) An indicator or parameter that shows the setting of a switch. (2) A character that signals the occurrence of some condition, such as the end of a word. (A) (3) (D of C) A characteristic of a file or directory that enables it to be used in certain ways. See also *archive flag*, *hidden flag*, and *read-only flag*.

focus - See *input focus*.

folder - A container used to organize objects.

font - A particular size and style of typeface that contains definitions of character sets, marker sets, and pattern sets.

Font Editor - A utility program provided with the IBM Developers Toolkit that enables the design and creation of new fonts.

foreground program - (1) The program with which the user is currently interacting. Also known as *interactive program*. Contrast with *background program*. (2) (D of C) In multiprogramming, a high-priority program.

frame - The part of a window that can contain several different visual elements specified by the application, but drawn and controlled by the Presentation Manager. The frame encloses the client area.

frame styles - Standard window layouts provided by the Presentation Manager.

FSD - File system driver.

full-duplex - Synonym for *duplex*.

full-screen application - An application that has complete control of the screen.

function - (1) In a programming language, a block, with or without formal parameters, whose execution is invoked by means of a call. (2) A set of related control statements that cause one or more programs to be performed.

function key - A key that causes a specified sequence of operations to be performed when it is pressed, for example, F1 and Alt-K.

function key area - The area at the bottom of a window that contains function key assignments such as F1=Help.

Glossary - G

GDT - Global Descriptor Table.

general protection fault - An exception condition that occurs when a process attempts to use storage or a module that has some level of protection assigned to it, such as I/O privilege level. See also *I/OPL code segment*.

Global Descriptor Table (GDT) - A table that defines code and data segments available to all tasks in an application.

global dynamic-link module - A dynamic-link module that can be shared by all processes in the system that refer to the module name.

global file-name character - Either a question mark (?) or an asterisk (*) used as a variable in a file name or file name extension when referring to a particular file or group of files.

glyph - A graphic symbol whose appearance conveys information.

GPI - Graphics programming interface.

graphic primitive - In computer graphics, a basic element, such as an arc or a line, that is not made up of smaller parts and that is used to create diagrams and pictures. See also *graphics segment*.

graphics - (1) A picture defined in terms of graphic primitives and graphics attributes. (2) (D of C) The making of charts and pictures. (3) Pertaining to charts, tables, and their creation. (4) See *computer graphics, coordinate graphics, fixed-image graphics, interactive graphics, passive graphics, raster graphics*.

graphics attributes - Attributes that apply to graphic primitives. Examples are color, line type, and shading-pattern definition. See also *segment attributes*.

graphics field - The clipping boundary that defines the visible part of the presentation-page contents.

graphics mode - One of several states of a display. The mode determines the resolution and color content of the screen.

graphics model space - The conceptual coordinate space in which a picture is constructed after any model transforms have been applied. Also known as *model space*.

Graphics programming interface - The formally defined programming language that is between an IBM graphics program and the user of the program.

graphics segment - A sequence of related graphic primitives and graphics attributes. See also *graphic primitive*.

graying - The indication that a choice on a pull-down is unavailable.

group - A collection of logically connected controls. For example, the buttons controlling paper size for a printer could be called a group. See also *program group*.

Glossary - H

handle - (1) An identifier that represents an object, such as a device or window, to the Presentation Interface. (2) (D of C) In the Advanced DOS and OS/2 operating systems, a binary value created by the system that identifies a drive, directory, and file so that the file can be found and opened.

hard error - An error condition on a network that requires either that the system be reconfigured or that the source of the error be removed before the system can resume reliable operation.

header - (1) System-defined control information that precedes user data. (2) The portion of a message that contains control information for the message, such as one or more destination fields, name of the originating station, input sequence number, character string indicating the type of message, and priority level for the message.

heading tags - A document element that enables information to be displayed in windows, and that controls entries in the contents window controls placement of push buttons in a window, and defines the shape and size of windows.

heap - An area of free storage available for dynamic allocation by an application. Its size varies according to the storage requirements of the application.

help function - (1) A function that provides information about a specific field, an application panel, or information about the help facility. (2) (D of C) One or more display images that describe how to use application software or how to do a system operation.

Help index - In SAA Common User Access architecture, a help action that provides an index of the help information available for an application.

help panel - A panel with information to assist users that is displayed in response to a help request from the user.

help window - A Common-User-Access-defined secondary window that displays information when the user requests help.

hidden file - An operating system file that is not displayed by a directory listing.

hide button - In the OS/2 operating system, a small, square button located in the right-hand corner of the title bar of a window that, when selected, removes from the screen all the windows associated with that window. Contrast with *maximize button*. See also *restore button*.

hierarchical inheritance - The relationship between parent and child classes. An object that is lower in the inheritance hierarchy than another object, inherits all the characteristics and behaviors of the objects above it in the hierarchy.

hierarchy - A tree of segments beginning with the root segment and proceeding downward to dependent segment types.

high-performance file system (HPFS) - In the OS/2 operating system, an installable file system that uses high-speed buffer storage, known as a cache, to provide fast access to large disk volumes. The file system also supports the coexistence of multiple, active file systems on a single personal computer, with the capability of multiple and different storage devices. File names used with the HPFS can have as many as 254 characters.

hit testing - The means of identifying which window is associated with which input device event.

hook - A point in a system-defined function where an application can supply additional code that the system processes as though it were part of the function.

hook chain - A sequence of hook procedures that are "chained" together so that each event is passed, in turn, to each procedure in the chain.

hot spot - The part of the pointer that must touch an object before it can be selected. This is usually the tip of the pointer. Contrast with *action point*.

HPFS - high-performance file system.

hypergraphic link - A connection between one piece of information and another through the use of graphics.

hypertext - A way of presenting information online with connections between one piece of information and another, called *hypertext links*. See also *hypertext link*.

hypertext link - A connection between one piece of information and another.

Glossary - I

I/O operation - An input operation to, or output operation from a device attached to a computer.

I-beam pointer - A pointer that indicates an area, such as an entry field in which text can be edited.

icon - In SAA Advanced Common User Access architecture, a graphical representation of an object, consisting of an image, image background, and a label. Icons can represent items (such as a document file) that the user wants to work on, and actions that the user wants to perform. In the Presentation Manager, icons are used for data objects, system actions, and minimized programs.

icon area - In the Presentation Manager, the area at the bottom of the screen that is normally used to display the icons for minimized programs.

Icon Editor - The Presentation Manager-provided tool for creating icons.

IDL - Interface Definition Language.

image font - A set of symbols, each of which is described in a rectangular array of pels. Some of the pels in the array are set to produce the image of one of the symbols. Contrast with *outline font*.

implied metaclass - Subclassing the metaclass of a parent class without a separate CSC for the resultant metaclass.

indirect manipulation - Interaction with an object through choices and controls.

information device context - A logical description of a data destination other than the screen (for example, a printer or plotter), but where no output will occur. Its purpose is to satisfy queries. See also *device context*.

information panel - A defined panel type characterized by a body containing only protected information.

Information Presentation Facility (IPF) - A facility provided by the OS/2 operating system, by which application developers can produce online documentation and context-sensitive online help panels for their applications.

inheritance - The technique of specifying the shape and behavior of one class (called a *subclass*) as incremental differences from another class (called the *parent class* or *superclass*). The subclass inherits the superclass' state representation and methods, and can provide additional data elements and methods. The subclass also can provide new functions with the same method names used by the superclass. Such a subclass method is said to override the superclass method, and will be selected automatically by method resolution on subclass instances. An overriding method can elect to call upon the superclass' method as part of its own implementation.

input focus - (1) The area of a window where user interaction is possible using an input device, such as a mouse or the keyboard. (2) The position in the *active window* where a user's normal interaction with the keyboard will appear.

input router - An internal OS/2 process that removes messages from the system queue.

input/output control - A device-specific command that requests a function of a device driver.

installable file system (IFS) - A file system in which software is installed when the operating system is started.

instance - (Or object instance). A specific object, as distinguished from the abstract definition of an object referred to as its class.

instance method - A method valid for a particular object.

instruction pointer - In System/38, a pointer that provides addressability for a machine interface instruction in a program.

integer atom - An *atom* that represents a predefined system constant and carries no storage overhead. For example, names of window classes provided by Presentation Manager are expressed as integer atoms.

interactive graphics - Graphics that can be moved or manipulated by a user at a terminal.

interactive program - (1) A program that is running (active) and is ready to receive (or is receiving) input from a user. (2) A running program that can receive input from the keyboard or another input device. Compare with *active program* and contrast with *noninteractive program*.

Also known as a *foreground program*.

interchange file - A file containing data that can be sent from one Presentation Manager interface application to another.

Interface Definition Language (IDL) - Language-neutral interface specification for a SOM class.

interpreter - A program that translates and executes each instruction of a high-level programming language before it translates and executes.

interprocess communication (IPC) - In the OS/2 operating system, the exchange of information between processes or threads through semaphores, pipes, queues, and shared memory.

interval timer - (1) A timer that provides program interruptions on a program-controlled basis. (2) An electronic counter that counts intervals of time under program control.

IOCtl - Input/output control.

IOPL - Input/output privilege level.

IOPL code segment - An IOPL executable section of programming code that enables an application to directly manipulate hardware interrupts and ports without replacing the device driver. See also *privilege level*.

IPC - Interprocess communication.

IPF - Information Presentation Facility.

IPF compiler - A text compiler that interpret tags in a source file and converts the information into the specified format.

IPF tag language - A markup language that provides the instructions for displaying online information.

item - A data object that can be passed in a DDE transaction.

Glossary - J

journal - A special-purpose file that is used to record changes made in the system.

Glossary - K

Kanji - A graphic character set used in Japanese ideographic alphabets.

KBD\$ - Character-device name reserved for the keyboard.

kernel - The part of an operating system that performs basic functions, such as allocating hardware resources.

kerning - The design of graphics characters so that their character boxes overlap. Used to space text proportionally.

keyboard accelerator - A keystroke that generates a command message for an application.

keyboard augmentation - A function that enables a user to press a keyboard key while pressing a mouse button.

keyboard focus - A temporary attribute of a window. The window that has a keyboard focus receives all keyboard input until the focus changes to a different window.

Keys help - In SAA Common User Access architecture, a help action that provides a listing of the application keys and their assigned functions.

Glossary - L

label - In a graphics segment, an identifier of one or more elements that is used when editing the segment.

LAN - Local area network.

language support procedure - A function provided by the Presentation Manager Interface for applications that do not, or cannot (as in the case of COBOL and FORTRAN programs), provide their own dialog or window procedures.

lazy drag - See *pickup and drop*.

lazy drag set - See *pickup set*.

LDT - In the OS/2 operating system, Local Descriptor Table.

LIFO stack - A stack from which data is retrieved in last-in, first-out order.

linear address - A unique value that identifies the memory object.

linked list - Synonym for *chained list*.

list box - In SAA Advanced Common User Access architecture, a control that contains scrollable choices from which a user can select one choice.

Note: In CUA architecture, this is a programmer term. The end user term is selection list.

list button - A button labeled with an underlined down-arrow that presents a list of valid objects or choices that can be selected for that field.

list panel - A defined panel type that displays a list of items from which users can select one or more choices and then specify one or more actions to work on those choices.

load time - The point in time at which a program module is loaded into main storage for execution.

load-on-call - A function of a linkage editor that allows selected segments of the module to be disk resident while other segments are executing. Disk resident segments are loaded for execution and given control when any entry point that they contain is called.

local area network (LAN) - (1) A computer network located on a user's premises within a limited geographical area. Communication within a local area network is not subject to external regulations; however, communication across the LAN boundary may be subject to some form of regulation. (T)

Note: A LAN does not use store and forward techniques. (2) A network in which a set of devices are connected to one another for communication and that can be connected to a larger network.

Local Descriptor Table (LDT) - Defines code and data segments specific to a single task.

lock - A serialization mechanism by means of which a resource is restricted for use by the holder of the lock.

logical storage device - A device that the user can map to a physical (actual) device.

LPT1, LPT2, LPT3 - Character-device names reserved for parallel printers 1 through 3.

Glossary - M

main window - The window that is positioned relative to the *desktop window*.

manipulation button - The button on a pointing device a user presses to directly manipulate an object.

map - (1) A set of values having a defined correspondence with the quantities or values of another set. (I) (A) (2) To establish a set of values having a defined correspondence with the quantities or values of another set. (I)

marker box - In computer graphics, the boundary that defines, in world coordinates, the horizontal and vertical space occupied by a single marker from a marker set.

marker symbol - A symbol centered on a point. Graphs and charts can use marker symbols to indicate the plotted points.

marquee box - The rectangle that appears during a selection technique in which a user selects objects by drawing a box around them with a pointing device.

Master Help Index - In the OS/2 operating system, an alphabetic list of help topics related to using the operating system.

maximize - To enlarge a window to its largest possible size.

media window - The part of the physical device (display, printer, or plotter) on which a picture is presented.

memory block - Part memory within a heap.

memory device context - A logical description of a data destination that is a memory bit map. See also *device context*.

memory management - A feature of the operating system for allocating, sharing, and freeing main storage.

memory object - Logical unit of memory requested by an application, which forms the granular unit of memory manipulation from the application viewpoint.

menu - In SAA Advanced Common User Access architecture, an extension of the menu bar that displays a list of choices available for a selected choice in the menu bar. After a user selects a choice in menu bar, the corresponding menu appears. Additional pop-up windows can appear from menu choices.

menu bar - In SAA Advanced Common User Access architecture, the area near the top of a window, below the title bar and above the rest of the window, that contains choices that provide access to other menus.

menu button - The button on a pointing device that a user presses to view a pop-up menu associated with an object.

message - (1) In the Presentation Manager, a packet of data used for communication between the Presentation Manager interface and Presentation Manager applications (2) In a user interface, information not requested by users but presented to users by the computer in response to a user action or internal process.

message box - (1) A dialog window predefined by the system and used as a simple interface for applications, without the necessity of creating dialog-template resources or dialog procedures. (2) (D of C) In SAA Advanced Common User Access architecture, a type of window that shows messages to users. See also *dialog box*, *primary window*, *secondary window*.

message filter - The means of selecting which messages from a specific window will be handled by the application.

message queue - A sequenced collection of messages to be read by the application.

message stream mode - A method of operation in which data is treated as a stream of messages. Contrast with *byte stream*.

metacharacter - See *global file-name character*.

metaclass - A class whose instances are all classes. In SOM, any class descended from SOMClass is a metaclass. The methods of a metaclass are sometimes called "class" methods.

metafile - A file containing a series of attributes that set color, shape and size, usually of a picture or a drawing. Using a program that can interpret these attributes, a user can view the assembled image.

metafile device context - A logical description of a data destination that is a metafile, which is used for graphics interchange. See also *device context*.

metalanguage - A language used to specify another language. For example, data types can be described using a metalanguage so as to make the descriptions independent of any one computer language.

method - One of the units that makes up the behavior of an object. A method is a combination of a function and a name, such that many different functions can have the same name. Which function the name refers to at any point in time depends on the object that is to execute the method and is the subject of method resolution.

method override - The replacement, by a child class, of the implementation of a method inherited from a parent and an ancestor class.

mickey - A unit of measurement for physical mouse motion whose value depends on the mouse device driver currently loaded.

micro presentation space - A graphics presentation space in which a restricted set of the GPI function calls is available.

minimize - To remove from the screen all windows associated with an application and replace them with an icon that represents the application.

mix - An attribute that determines how the foreground of a graphic primitive is combined with the existing color of graphics output. Also known as *foreground mix*. Contrast with *background mix*.

mixed character string - A string containing a mixture of one-byte and *Kanji* or Hangeul (two-byte) characters.

mnemonic - (1) A method of selecting an item on a pull-down by means of typing the highlighted letter in the menu item. (2) (D of C) In SAA Advanced Common User Access architecture, usually a single character, within the text of a choice, identified by an underscore beneath the character. If all characters in a choice already serve as mnemonics for other choices, another character, placed in parentheses immediately following the choice, can be used. When a user types the mnemonic for a choice, the choice is either selected or the cursor is moved to that choice.

modal dialog box - In SAA Advanced Common User Access architecture, a type of movable window, fixed in size, that requires a user to enter information before continuing to work in the application window from which it was displayed. Contrast with *modeless dialog box*. Also known as a *serial dialog box*. Contrast with *parallel dialog box*.

Note: In CUA architecture, this is a programmer term. The end user term is pop-up window.

model space - See *graphics model space*.

modeless dialog box - In SAA Advanced Common User Access architecture, a type of movable window, fixed in size, that allows users to continue their dialog with the application without entering information in the dialog box. Also known as a *parallel dialog box*. Contrast with *modal dialog box*.

Note: In CUA architecture, this is a programmer term. The end user term is pop-up window.

module definition file - A file that describes the code segments within a load module. For example, it indicates whether a code segment is loadable before module execution begins (preload), or loadable only when referred to at run time (load-on-call).

mouse - In SAA usage, a device that a user moves on a flat surface to position a pointer on the screen. It allows a user to select a choice or function to be performed or to perform operations on the screen, such as dragging or drawing lines from one position to another.

MOUSE\$ - Character-device name reserved for a mouse.

multiple-choice selection - In SAA Basic Common User Access architecture, a type of field from which a user can select one or more choices or select none. See also *check box*. Contrast with *extended-choice selection*.

multiple-line entry field - In SAA Advanced Common User Access architecture, a control into which a user types more than one line of information. See also *single-line entry field*.

multitasking - The concurrent processing of applications or parts of applications. A running application and its data are protected from other concurrently running applications.

mutex semaphore - (Mutual exclusion semaphore). A semaphore that enables threads to serialize their access to resources. Only the thread that currently owns the mutex semaphore can gain access to the resource, thus preventing one thread from interrupting operations being performed by another.

muxwait semaphore - (Multiple wait semaphore). A semaphore that enables a thread to wait either for multiple event semaphores to be posted or for multiple mutex semaphores to be released. Alternatively, a muxwait semaphore can be set to enable a thread to wait for any ONE of the event or mutex semaphores in the muxwait semaphore's list to be posted or released.

Glossary - N

named pipe - A named buffer that provides client-to-server, server-to-client, or full duplex communication between unrelated processes. Contrast with *unnamed pipe*.

national language support (NLS) - The modification or conversion of a United States English product to conform to the requirements of another language or country. This can include the enabling or retrofitting of a product and the translation of nomenclature, MRI, or documentation of a product.

nested list - A list that is contained within another list.

NLS - national language support.

non-8.3 file-name format - A file-naming convention in which file names can consist of up to 255 characters. See also *8.3 file-name format*.

noncritical extended attribute - An extended attribute that is not necessary for the function of an application.

nondestructive read - Reading that does not erase the data in the source location. (T)

noninteractive program - A running program that cannot receive input from the keyboard or other input device. Compare with *active program*, and contrast with *interactive program*.

nonretained graphics - Graphic primitives that are not remembered by the Presentation Manager interface when they have been drawn. Contrast with *retained graphics*.

null character (NUL) - (1) Character-device name reserved for a nonexistent (dummy) device. (2) (D of C) A control character that is used to accomplish media-fill or time-fill and that may be inserted into or removed from a sequence of characters without affecting the meaning of the sequence; however, the control of equipment or the format may be affected by this character. (I) (A)

null-terminated string - A string of (n+1) characters where the (n+1)th character is the 'null' character (0x00) Also known as

'zero-terminated' string and 'ASCII' string.

Glossary - O

object - The elements of data and function that programs create, manipulate, pass as arguments, and so forth. An object is a way of associating specific data values with a specific set of named functions (called *methods*) for a period of time (referred to as the *lifetime* of the object). The data values of an object are referred to as its *state*. In SOM, objects are created by other objects called *classes*. The specification of what comprises the set of functions and data elements that make up an object is referred to as the *definition* of a class.

SOM objects offer a high degree of *encapsulation*. This property permits many aspects of the implementation of an object to change without affecting client programs that depend on the object's behavior.

object definition - See *class*.

object instance - See *instance*.

Object Interface Definition Language (OIDL) - Specification language used in SOM Version 1 for defining classes. Replaced by Interface Definition Language (IDL).

object window - A window that does not have a parent but which might have child windows. An object window cannot be presented on a device.

OIDL - Object Interface Definition Language.

open - To start working with a file, directory, or other object.

ordered list - Vertical arrangements of items, with each item in the list preceded by a number or letter.

outline font - A set of symbols, each of which is created as a series of lines and curves. Synonymous with *vector font*. Contrast with *image font*.

output area - An area of storage reserved for output. (A)

owner window - A window into which specific events that occur in another (owned) window are reported.

ownership - The determination of how windows communicate using messages.

owning process - The process that owns the resources that might be shared with other processes.

Glossary - P

page - (1) A 4KB segment of contiguous physical memory. (2) (D of C) A defined unit of space on a storage medium.

page viewport - A boundary in device coordinates that defines the area of the output device in which graphics are to be displayed. The presentation-page contents are transformed automatically to the page viewport in device space.

paint - (1) The action of drawing or redrawing the contents of a window. (2) In computer graphics, to shade an area of a display image; for example, with crosshatching or color.

panel - In SAA Basic Common User Access architecture, a particular arrangement of information that is presented in a window or pop-up. If some of the information is not visible, a user can scroll through the information.

panel area - An area within a panel that contains related information. The three major Common User Access-defined panel areas are the action bar, the function key area, and the panel body.

panel area separator - In SAA Basic Common User Access architecture, a solid, dashed, or blank line that provides a visual distinction between two adjacent areas of a panel.

panel body - The portion of a panel not occupied by the action bar, function key area, title or scroll bars. The panel body can contain protected information, selection fields, and entry fields. The layout and content of the panel body determine the panel type.

panel body area - See *client area*.

panel definition - A description of the contents and characteristics of a panel. A panel definition is the application developer's mechanism for predefining the format to be presented to users in a window.

panel ID - In SAA Basic Common User Access architecture, a panel identifier, located in the upper-left corner of a panel. A user can choose whether to display the panel ID.

panel title - In SAA Basic Common User Access architecture, a particular arrangement of information that is presented in a window or pop-up. If some of the information is not visible, a user can scroll through the information.

paper size - The size of paper, defined in either standard U.S. or European names (for example, A, B, A4), and measured in inches or millimeters respectively.

parallel dialog box - See *modeless dialog box*.

parameter list - A list of values that provides a means of associating addressability of data defined in a called program with data in the calling program. It contains parameter names and the order in which they are to be associated in the calling and called program.

parent class - See *inheritance*.

parent process - In the OS/2 operating system, a process that creates other processes. Contrast with *child process*.

parent window - In the OS/2 operating system, a window that creates a child window. The child window is drawn within the parent window. If the parent window is moved, resized, or destroyed, the child window also will be moved, resized, or destroyed. However, the child window can be moved and resized independently from the parent window, within the boundaries of the parent window. Contrast with *child window*.

partition - (1) A fixed-size division of storage. (2) On an IBM personal computer fixed disk, one of four possible storage areas of variable size; one may be accessed by DOS, and each of the others may be assigned to another operating system.

Paste - A choice in the Edit pull-down that a user selects to move the contents of the clipboard into a preselected location. See also *Copy* and *Cut*.

path - The route used to locate files; the storage location of a file. A fully qualified path lists the drive identifier, directory name, subdirectory name (if any), and file name with the associated extension.

PDD - Physical device driver.

peeking - An action taken by any thread in the process that owns the queue to examine queue elements without removing them.

pel - (1) The smallest area of a display screen capable of being addressed and switched between visible and invisible states. Synonym for *display point*, *pixel*, and *picture element*. (2) (D of C) Picture element.

persistent object - An object whose instance data and state are preserved between system shutdown and system startup.

physical device driver (PDD) - A system interface that handles hardware interrupts and supports a set of input and output functions.

pick - To select part of a displayed object using the pointer.

pickup - To add an object or set of objects to the pickup set.

pickup and drop - A drag operation that does not require the direct manipulation button to be pressed for the duration of the drag.

pickup set - The set of objects that have been picked up as part of a pickup and drop operation.

picture chain - See *segment chain*.

picture element - (1) Synonym for *pel*. (2) (D of C) In computer graphics, the smallest element of a display surface that can be independently assigned color and intensity. (T) . (3) The area of the finest detail that can be reproduced effectively on the recording medium.

PID - Process identification.

pipe - (1) A named or unnamed buffer used to pass data between processes. A process reads from or writes to a pipe as if the pipe were a standard-input or standard-output file. See also *named pipe* and *unnamed pipe*. (2) (D of C) To direct data so that the output from one process becomes the input to another process. The standard output of one command can be connected to the standard input of another with the pipe operator (|).

pixel - (1) Synonym for *pel*. (2) (D of C) Picture element.

plotter - An output unit that directly produces a hardcopy record of data on a removable medium, in the form of a two-dimensional graphic representation. (T)

PM - Presentation Manager.

pointer - (1) The symbol displayed on the screen that is moved by a pointing device, such as a *mouse*. The pointer is used to point at items that users can select. Contrast with *cursor*. (2) A data element that indicates the location of another data element. (T)

POINTER\$ - Character-device name reserved for a pointer device (mouse screen support).

pointing device - In SAA Advanced Common User Access architecture, an instrument, such as a mouse, trackball, or joystick, used to move a pointer on the screen.

pointings - Pairs of x-y coordinates produced by an operator defining positions on a screen with a pointing device, such as a *mouse*.

polyfillet - A curve based on a sequence of lines. The curve is tangential to the end points of the first and last lines, and tangential also to the midpoints of all other lines. See also *fillet*.

polygon - One or more closed figures that can be drawn filled, outlined, or filled and outlined.

polyline - A sequence of adjoining lines.

polymorphism - The ability to have different implementations of the same method for two or more classes of objects.

pop - To retrieve an item from a last-in-first-out stack of items. Contrast with *push*.

pop-up menu - A menu that lists the actions that a user can perform on an object. The contents of the pop-up menu can vary depending on the context, or state, of the object.

pop-up window - (1) A window that appears on top of another window in a dialog. Each pop-up window must be completed before returning to the underlying window. (2) (D of C) In SAA Advanced Common User Access architecture, a movable window, fixed in size, in which a user provides information required by an application so that it can continue to process a user request.

presentation drivers - Special purpose I/O routines that handle field device-independent I/O requests from the PM and its applications.

Presentation Manager (PM) - The interface of the OS/2 operating system that presents, in windows a graphics-based interface to applications and files installed and running under the OS/2 operating system.

presentation page - The coordinate space in which a picture is assembled for display.

presentation space (PS) - (1) Contains the device-independent definition of a picture. (2) (D of C) The display space on a display device.

primary window - In SAA Common User Access architecture, the window in which the main interaction between the user and the application takes place. In a multiprogramming environment, each application starts in its own primary window. The primary window remains for the duration of the application, although the panel displayed will change as the user's dialog moves forward. See also *secondary window*.

primitive - In computer graphics, one of several simple functions for drawing on the screen, including, for example, the rectangle, line, ellipse, polygon, and so on.

primitive attribute - A specifiable characteristic of a graphic primitive. See *graphics attributes*.

print job - The result of sending a document or picture to be printed.

Print Manager - In the Presentation Manager, the part of the spooler that manages the spooling process. It also allows users to view print queues and to manipulate print jobs.

privilege level - A protection level imposed by the hardware architecture of the IBM personal computer. There are four privilege levels (number 0 through 3). Only certain types of programs are allowed to execute at each privilege level. See also *IOPL code segment*.

procedure call - In programming languages, a language construct for invoking execution of a procedure.

process - An instance of an executing application and the resources it is using.

program - A sequence of instructions that a computer can interpret and execute.

program details - Information about a program that is specified in the *Program Manager* window and is used when the program is started.

program group - In the Presentation Manager, several programs that can be acted upon as a single entity.

program name - The full file specification of a program. Contrast with *program title*.

program title - The name of a program as it is listed in the *Program Manager* window. Contrast with *program name*.

prompt - A displayed symbol or message that requests input from the user or gives operational information; for example, on the display screen of an IBM personal computer, the DOS A> prompt. The user must respond to the prompt in order to proceed.

protect mode - A method of program operation that limits or prevents access to certain instructions or areas of storage. Contrast with *real mode*.

protocol - A set of semantic and syntactic rules that determines the behavior of functional units in achieving communication. (I)

pseudocode - An artificial language used to describe computer program algorithms without using the syntax of any particular programming language. (A)

pull-down - (1) An *action bar* extension that displays a list of choices available for a selected action bar choice. After users select an action bar choice, the pull-down appears with the list of choices. Additional *pop-up windows* may appear from pull-down choices to further extend the actions available to users. (2) (D of C) In SAA Common User Access architecture, pertaining to a choice in an action bar pull-down.

push - To add an item to a last-in-first-out stack of items. Contrast with *pop*.

push button - In SAA Advanced Common User Access architecture, a rectangle with text inside. Push buttons are used in windows for actions that occur immediately when the push button is selected.

putback - To remove an object or set of objects from the lazy drag set. This has the effect of undoing the pickup operation for those objects

putdown - To drop the objects in the lazy drag set on the target object.

Glossary - Q

queue - (1) A linked list of elements waiting to be processed in FIFO order. For example, a queue may be a list of print jobs waiting to be printed. (2) (D of C) A line or list of items waiting to be processed; for example, work to be performed or messages to be displayed.

queued device context - A logical description of a data destination (for example, a printer or plotter) where the output is to go through the spooler. See also *device context*.

Glossary - R

radio button - (1) A control window, shaped like a round button on the screen, that can be in a checked or unchecked state. It is used to select a single item from a list. Contrast with *check box*. (2) In SAA Advanced Common User Access architecture, a circle with text beside it. Radio buttons are combined to show a user a fixed set of choices from which only one can be selected. The circle is partially filled when a choice is selected.

RAS - Reliability, availability, and serviceability.

raster - (1) In computer graphics, a predetermined pattern of lines that provides uniform coverage of a display space. (T) (2) The coordinate grid that divides the display area of a display device. (A)

read-only file - A file that can be read from but not written to.

real mode - A method of program operation that does not limit or prevent access to any instructions or areas of storage. The operating system loads the entire program into storage and gives the program access to all system resources. Contrast with *protect mode*.

realize - To cause the system to ensure, wherever possible, that the physical color table of a device is set to the closest possible match in the logical color table.

recursive routine - A routine that can call itself, or be called by another routine that was called by the recursive routine.

reentrant - The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks.

reference phrase - (1) A word or phrase that is emphasized in a device-dependent manner to inform the user that additional information for the word or phrase is available. (2) (D of C) In hypertext, text that is highlighted and preceded by a single-character input field used to signify the existence of a hypertext link.

reference phrase help - In SAA Common User Access architecture, highlighted words or phrases within help information that a user selects to get additional information.

refresh - To update a window, with changed information, to its current status.

region - A clipping boundary in device space.

register - A part of internal storage having a specified storage capacity and usually intended for a specific purpose. (T)

remote file system - A file-system driver that gains access to a remote system without a block device driver.

resource - The means of providing extra information used in the definition of a window. A resource can contain definitions of fonts, templates, accelerators, and mnemonics; the definitions are held in a resource file.

resource file - A file containing information used in the definition of a window. Definitions can be of fonts, templates, accelerators, and mnemonics.

restore - To return a window to its original size or position following a sizing or moving action.

retained graphics - Graphic primitives that are remembered by the Presentation Manager interface after they have been drawn. Contrast with *nonretained graphics*.

return code - (1) A value returned to a program to indicate the results of an operation requested by that program. (2) A code used to influence the execution of succeeding instructions.(A)

reverse video - (1) A form of highlighting a character, field, or cursor by reversing the color of the character, field, or cursor with its background; for example, changing a red character on a black background to a black character on a red background. (2) In SAA Basic Common User Access architecture, a screen emphasis feature that interchanges the foreground and background colors of an item.

REXX Language - Restructured Extended Executor. A procedural language that provides batch language functions along with structured programming constructs such as loops; conditional testing and subroutines.

RGB - (1) Color coding in which the brightness of the additive primary colors of light, red, green, and blue, are specified as three distinct values of white light. (2) Pertaining to a color display that accepts signals representing red, green, and blue.

roman - Relating to a type style with upright characters.

root segment - In a hierarchical database, the highest segment in the tree structure.

round-robin scheduling - A process that allows each thread to run for a specified amount of time.

run time - (1) Any instant at which the execution of a particular computer program takes place. (T) (2) The amount of time needed for the execution of a particular computer program. (T) (3) The time during which an instruction in an instruction register is decoded and performed. Synonym for *execution time*.

Glossary - S

SAA - Systems Application Architecture.

SBCS - Single-byte character set.

scheduler - A computer program designed to perform functions such as scheduling, initiation, and termination of jobs.

screen - In SAA Basic Common User Access architecture, the physical surface of a display device upon which information is shown to a user.

screen device context - A logical description of a data destination that is a particular window on the screen. See also *device context*.

SCREEN\$ - Character-device name reserved for the display screen.

scroll bar - In SAA Advanced Common User Access architecture, a part of a window, associated with a scrollable area, that a user interacts with to see information that is not currently visible.

scrollable entry field - An entry field larger than the visible field.

scrollable selection field - A selection field that contains more choices than are visible.

scrolling - Moving a display image vertically or horizontally in a manner such that new data appears at one edge, as existing data disappears at the opposite edge.

secondary window - A window that contains information that is dependent on information in a primary window and is used to supplement the interaction in the primary window.

sector - On disk or diskette storage, an addressable subdivision of a track used to record one block of a program or data.

segment - See *graphics segment*.

segment attributes - Attributes that apply to the segment as an entity, as opposed to the individual primitives within the segment. For example, the visibility or detectability of a segment.

segment chain - All segments in a graphics presentation space that are defined with the 'chained' attribute. Synonym for *picture chain*.

segment priority - The order in which segments are drawn.

segment store - An area in a normal graphics presentation space where retained graphics segments are stored.

select - To mark or choose an item. Note that *select* means to mark or type in a choice on the screen; *enter* means to send all selected choices to the computer for processing.

select button - The button on a pointing device, such as a mouse, that is pressed to select a menu choice. Also known as button 1.

selection cursor - In SAA Advanced Common User Access architecture, a visual indication that a user has selected a choice. It is represented by outlining the choice with a dotted box. See also *text cursor*.

selection field - (1) In SAA Advanced Common User Access architecture, a set of related choices. See also *entry field*. (2) In SAA Basic Common User Access architecture, an area of a panel that cannot be scrolled and contains a fixed number of choices.

semantics - The relationships between symbols and their meanings.

semaphore - An object used by applications for signalling purposes and for controlling access to serially reusable resources.

separator - In SAA Advanced Common User Access architecture, a line or color boundary that provides a visual distinction between two adjacent areas.

serial dialog box - See *modal dialog box*.

serialization - The consecutive ordering of items.

serialize - To ensure that one or more events occur in a specified sequence.

serially reusable resource (SRR) - A logical resource or object that can be accessed by only one task at a time.

session - (1) A routing mechanism for user interaction via the console; a complete environment that determines how an application runs and how users interact with the application. OS/2 can manage more than one session at a time, and more than one process can run in a session. Each session has its own set of environment variables that determine where OS/2 looks for dynamic-link libraries and other important files. (2) (D of C) In the OS/2 operating system, one instance of a started program or command prompt. Each session is separate from all other sessions that might be running on the computer. The operating system is responsible for coordinating the resources that each session uses, such as computer memory, allocation of processor time, and windows on the screen.

Settings Notebook - A control window that is used to display the settings for an object and to enable the user to change them.

shadow - An object that refers to another object. A shadow is not a copy of another object, but is another representation of the object.

shadow box - The area on the screen that follows mouse movements and shows what shape the window will take if the mouse button is released.

shared data - Data that is used by two or more programs.

shared memory - In the OS/2 operating system, a segment that can be used by more than one program.

shear - In computer graphics, the forward or backward slant of a graphics symbol or string of such symbols relative to a line perpendicular to

the baseline of the symbol.

shell - (1) A software interface between a user and the operating system of a computer. Shell programs interpret commands and user interactions on devices such as keyboards, pointing devices, and touch-sensitive screens, and communicate them to the operating system. (2) Software that allows a kernel program to run under different operating-system environments.

shutdown - The process of ending operation of a system or a subsystem, following a defined procedure.

sibling processes - Child processes that have the same parent process.

sibling windows - Child windows that have the same parent window.

simple list - A list of like values; for example, a list of user names. Contrast with *mixed list*.

single-byte character set (SBCS) - A character set in which each character is represented by a one-byte code. Contrast with *double-byte character set*.

slider box - In SAA Advanced Common User Access architecture: a part of the scroll bar that shows the position and size of the visible information in a window relative to the total amount of information available. Also known as *thumb mark*.

SOM - System Object Model.

source file - A file that contains source statements for items such as high-level language programs and data description specifications.

source statement - A statement written in a programming language.

specific dynamic-link module - A dynamic-link module created for the exclusive use of an application.

spin button - In SAA Advanced Common User Access architecture, a type of entry field that shows a scrollable ring of choices from which a user can select a choice. After the last choice is displayed, the first choice is displayed again. A user can also type a choice from the scrollable ring into the entry field without interacting with the spin button.

spline - A sequence of one or more Bézier curves.

spooler - A program that intercepts the data going to printer devices and writes it to disk. The data is printed or plotted when it is complete and the required device is available. The spooler prevents output from different sources from being intermixed.

stack - A list constructed and maintained so that the next data element to be retrieved is the most recently stored. This method is characterized as last-in-first-out (LIFO).

standard window - A collection of window elements that form a panel. The standard window can include one or more of the following window elements: sizing borders, system menu icon, title bar, maximize/minimize/restore icons, action bar and pull-downs, scroll bars, and client area.

static control - The means by which the application presents descriptive information (for example, headings and descriptors) to the user. The user cannot change this information.

static storage - (1) A read/write storage unit in which data is retained in the absence of control signals. (A) Static storage may use dynamic addressing or sensing circuits. (2) Storage other than *dynamic storage*. (A)

style - See *window style*.

subclass - A class that inherits from another class. See also *Inheritance*.

subdirectory - In an IBM personal computer, a file referred to in a root directory that contains the names of other files stored on the diskette or fixed disk.

superclass - A class from which another class inherits. See also *inheritance*.

swapping - (1) A process that interchanges the contents of an area of real storage with the contents of an area in auxiliary storage. (I) (A) (2) In a system with virtual storage, a paging technique that writes the active pages of a job to auxiliary storage and reads pages of another job from auxiliary storage into real storage. (3) The process of temporarily removing an active job from main storage, saving it on disk, and processing another job in the area of main storage formerly occupied by the first job.

switch - (1) In SAA usage, to move the cursor from one point of interest to another; for example, to move from one screen or window to another or from a place within a displayed image to another place on the same displayed image. (2) In a computer program, a conditional instruction and an indicator to be interrogated by that instruction. (3) A device or programming technique for making a selection, for example, a toggle, a conditional jump.

switch list - See *Task List*.

symbolic identifier - A text string that equates to an integer value in an include file, which is used to identify a programming object.

symbols - In Information Presentation Facility, a document element used to produce characters that cannot be entered from the keyboard.

synchronous - Pertaining to two or more processes that depend upon the occurrence of specific events such as common timing signals. (T)
See also *asynchronous*.

System Menu - In the Presentation Manager, the pull-down in the top left corner of a window that allows it to be moved and sized with the keyboard.

System Object Model (SOM) - A mechanism for language-neutral, object-oriented programming in the OS/2 environment.

system queue - The master queue for all pointer device or keyboard events.

system-defined messages - Messages that control the operations of applications and provides input and other information for applications to process.

Systems Application Architecture (SAA) - A set of IBM software interfaces, conventions, and protocols that provide a framework for designing and developing applications that are consistent across systems.

Glossary - T

table tags - In Information Presentation Facility, a document element that formats text in an arrangement of rows and columns.

tag - (1) One or more characters attached to a set of data that contain information about the set, including its identification. (I) (A) (2) In Generalized Markup Language markup, a name for a type of document or document element that is entered in the source document to identify it.

target object - An object to which the user is transferring information.

Task List - In the Presentation Manager, the list of programs that are active. The list can be used to switch to a program and to stop programs.

terminate-and-stay-resident (TSR) - Pertaining to an application that modifies an operating system interrupt vector to point to its own location (known as hooking an interrupt).

text - Characters or symbols.

text cursor - A symbol displayed in an entry field that indicates where typed input will appear.

text window - Also known as the VIO window.

text-windowed application - The environment in which the operating system performs advanced-video input and output operations.

thread - A unit of execution within a process. It uses the resources of the process.

thumb mark - The portion of the scroll bar that describes the range and properties of the data that is currently visible in a window. Also known as a *slider box*.

thunk - Term used to describe the process of address conversion, stack and structure realignment, etc., necessary when passing control between 16-bit and 32-bit modules.

tilde - A mark used to denote the character that is to be used as a mnemonic when selecting text items within a menu.

time slice - (1) An interval of time on the processing unit allocated for use in performing a task. After the interval has expired, processing-unit time is allocated to another task, so a task cannot monopolize processing-unit time beyond a fixed limit. (2) In systems with time sharing, a segment of time allocated to a terminal job.

time-critical process - A process that must be performed within a specified time after an event has occurred.

timer - A facility provided under the Presentation Manager, whereby Presentation Manager will dispatch a message of class WM_TIMER to a particular window at specified intervals. This capability may be used by an application to perform a specific processing task at predetermined intervals, without the necessity for the application to explicitly keep track of the passage of time.

timer tick - See *clock tick*.

title bar - In SAA Advanced Common User Access architecture, the area at the top of each window that contains the window title and

system menu icon. When appropriate, it also contains the minimize, maximize, and restore icons. Contrast with *panel title*.

TLB - Translation lookaside buffer.

transaction - An exchange between a workstation and another device that accomplishes a particular action or result.

transform - (1) The action of modifying a picture by scaling, shearing, reflecting, rotating, or translating. (2) The object that performs or defines such a modification; also referred to as a *transformation*.

Translation lookaside buffer (TLB) - A hardware-based address caching mechanism for paging information.

Tree - In the Presentation Manager, the window in the *File Manager* that shows the organization of drives and directories.

truncate - (1) To terminate a computational process in accordance with some rule (A) (2) To remove the beginning or ending elements of a string. (3) To drop data that cannot be printed or displayed in the line width specified or available. (4) To shorten a field or statement to a specified length.

TSR - Terminate-and-stay-resident.

Glossary - U

unnamed pipe - A circular buffer, created in memory, used by related processes to communicate with one another. Contrast with *named pipe*.

unordered list - In Information Presentation Facility, a vertical arrangement of items in a list, with each item in the list preceded by a special character or bullet.

update region - A system-provided area of dynamic storage containing one or more (not necessarily contiguous) rectangular areas of a window that are visually invalid or incorrect, and therefore are in need of repainting.

user interface - Hardware, software, or both that allows a user to interact with and perform operations on a system, program, or device.

User Shell - A component of OS/2 that uses a graphics-based, windowed interface to allow the user to manage applications and files installed and running under OS/2.

utility program - (1) A computer program in general support of computer processes; for example, a diagnostic program, a trace program, a sort program. (T) (2) A program designed to perform an everyday task such as copying data from one storage device to another. (A)

Glossary - V

value set control - A visual component that enables a user to select one choice from a group of mutually exclusive choices.

vector font - A set of symbols, each of which is created as a series of lines and curves. Synonymous with *outline font*. Contrast with *image font*.

VGA - Video graphics array.

view - A way of looking at an object's information.

viewing pipeline - The series of transformations applied to a graphic object to map the object to the device on which it is to be presented.

viewing window - A clipping boundary that defines the visible part of model space.

VIO - Video Input/Output.

virtual memory (VM) - Synonymous with *virtual storage*.

virtual storage - (1) The storage space that may be regarded as addressable main storage by the user of a computer system in which

virtual addresses are mapped into real addresses. The size of virtual storage is limited by the addressing scheme of the computer system and by the amount of auxiliary storage available, not by the actual number of main storage locations. (1) (A) (2) Addressable space that is apparent to the user as the processor storage space, from which the instructions and the data are mapped into the processor storage locations. (3) Synonymous with *virtual memory*.

visible region - A window's presentation space, clipped to the boundary of the window and the boundaries of any overlying window.

volume - (1) A file-system driver that uses a block device driver for input and output operations to a local or remote device. (1) (2) A portion of data, together with its data carrier, that can be handled conveniently as a unit.

Glossary - W

wildcard character - Synonymous with *global file-name character*.

window - (1) A portion of a display surface in which display images pertaining to a particular application can be presented. Different applications can be displayed simultaneously in different windows. (A) (2) An area of the screen with visible boundaries within which information is displayed. A window can be smaller than or the same size as the screen. Windows can appear to overlap on the screen.

window class - The grouping of windows whose processing needs conform to the services provided by one window procedure.

window coordinates - A set of coordinates by which a window position or size is defined; measured in device units, or *pels*.

window handle - Unique identifier of a window, generated by Presentation Manager when the window is created, and used by applications to direct messages to the window.

window procedure - Code that is activated in response to a message. The procedure controls the appearance and behavior of its associated windows.

window rectangle - The means by which the size and position of a window is described in relation to the desktop window.

window resource - A read-only data segment stored in the .EXE file of an application or the .DLL file of a dynamic link library.

window style - The set of properties that influence how events related to a particular window will be processed.

window title - In SAA Advanced Common User Access architecture, the area in the title bar that contains the name of the application and the OS/2 operating system file name, if applicable.

Workplace Shell - The OS/2 object-oriented, graphical user interface.

workstation - (1) A display screen together with attachments such as a keyboard, a local copy device, or a tablet. (2) (D of C) One or more programmable or nonprogrammable devices that allow a user to do work.

world coordinates - A device-independent Cartesian coordinate system used by the application program for specifying graphical input and output. (1) (A)

world-coordinate space - Coordinate space in which graphics are defined before transformations are applied.

WYSIWYG - What-You-See-Is-What-You-Get. A capability of a text editor to continually display pages exactly as they will be printed.

Glossary - X

There are no glossary terms for this starting letter.

Glossary - Y

There are no glossary terms for this starting letter.

Glossary - Z

z-order - The order in which sibling windows are presented. The topmost sibling window obscures any portion of the siblings that it overlaps; the same effect occurs down through the order of lower sibling windows.

zooming - The progressive scaling of an entire display image in order to give the visual impression of movement of all or part of a display group toward or away from an observer. (I) (A)

8.3 file-name format - A file-naming convention in which file names are limited to eight characters before and three characters after a single dot. Usually pronounced "eight-dot-three." See also *non-8.3 file-name format*.
